MTR-4723
VOL. I

JSC #14792

JUL 3 1979

NASA CR-
160451

TMS Communications Software
Volume I- Computer Interfaces

J. S. Brown
M. D. Lenker

APRIL 1979

MITRE

MITRE Technical Report

MTR-4723

Vol. I

TMS Communications Software
Volume I- Computer Interfaces

J. S. Brown
M. D. Lenker

APRIL 1979

CONTRACT SPONSOR	NASA/JSC
CONTRACT NO.	F19628-79-C-0001 T5295F
PROJECT NO.	8470
DEPT.	D72

HOUSTON, TEXAS

Department Approval: Edwin S. Kenda

MITRE Project Approval: Edwin S. Kenda

ABSTRACT

At NASA's Johnson Space Center MITRE has installed a prototype bus communications system, which is being used to support the Trend Monitoring System (TMS) as well as for evaluation of the bus concept. As a part of the TMS bus installation, MITRE implemented hardware and software interfaces to the MODCOMP and NOVA minicomputers included in the system. This document describes the system software required to drive the interfaces in each TMS computer. Documentation of other software for bus statistics monitoring and for transferring files across the bus is also included.

TABLE OF CONTENTS

PAGE IV INTENTIONALLY BLANK

LIST OF ILLUSTRATIONS

LIST OF TABLES

TREND MONITORING SYSTEM (TMS)
COMMUNICATIONS SOFTWARE
VOLUME I
COMPUTER INTERFACES

SECTION I
INTRODUCTION

1.0 BACKGROUND

The Orbiter Data Reduction Complex (ODRC) at NASA's Johnson Space Center has the responsibility of providing data reduction for measurements collected during manned spaceflight missions. This data reduction involves the extraction of requested data from magnetic tapes, the calibration of the raw measurements and the conversion of the measurements to engineering units, and the display of the data in any of a variety of output forms. Ordinarily, the work of the ODRC is done in response to written requests and has a planned turnaround time ranging from several hours to several days, depending on the priority of the request.

In 1977, however, as data processing requirements for Operational Flight Tests (OFT) of the Space Shuttle were being considered, it was established that NASA/JSC's Structures and Mechanics Division (SMD) needed to view thermal parameters for the shuttle in near real time. As a consequence, the Institutional Data Systems Division (IDSD), which is responsible for the ODRC, chose to implement an interactive graphics system to display plots of current, projected, and historical thermal data for the Shuttle. The system, termed the Trend Monitoring System (TMS), was implemented by IDSD's Engineering and Special Development Branch (FD7) using a MODCOMP IV/35 host minicomputer and MEGATEK 5000 intelligent graphics terminals (based around Data General NOVA/3 minicomputers).

In the TMS, the terminals and the host computer are separated by a distance of about 1600 feet, and the requirements for response time dictate that a high data rate be provided on the communications path between the host and the terminals. Conventional communications systems to meet these

requirements are not readily available. MITRE has developed a coaxial cable bus communications system [1] which provides a communications bandwidth of up to 307.2 Kbps over a distance of several miles. IDSD consequently elected to install a prototype bus communications system with the dual objective of supporting the TMS needs and of providing a test bed for further evaluation of the bus concept's ability to meet digital computer communication needs.

MITRE has provided both engineering and implementation support for the prototype bus. This work is documented in a series of reports ([2], [3], [4], [5], [6], [7]).

1.1 Overview of This Report

In the NASA bus system, subscriber devices (terminals or computers) are interfaced to the bus through microprocessor-based Bus Interface Units (BIUs). As a part of the prototype installation, MITRE implemented hardware and software interfaces between the NOVAs and their BIUs and between the MODCOMP and its BIU. In each case, system software is required to deal with the interface.

This report documents the interface software in both the NOVA and the MODCOMP (Software written for the BIUs themselves is documented in [6]). Section II deals with the MODCOMP symbiont, which provides queuing, logical-to-physical device mapping, and I/O handler control. This section also describes two bus-connected support programs -- the routines to report statistics about bus usage, and the MODCOMP program used in transferring a program file from the NOVA to the MODCOMP. The file transfer process is needed because software for the intelligent terminals is developed on a MEGATEK terminal augmented with two floppy disk drives. The absolute code version of the terminal software is then transferred back to the MODCOMP. From the MODCOMP, the intelligent terminals can be bootstrapped over the communications medium on demend. The MODCOMP must be involved in the bootstrapping because only one of the MEGATEKs -- the development terminal -- has a nonvolatile storage medium.

Section III of this report documents the I/O handler developed for the NOVAs and also the NOVA program used in transferring a program file from the NOVA to the MODCOMP.

SECTION II
MODCOMP SOFTWARE

2.0 INTRODUCTION

This section provides information needed to understand the background, design, and structure of the MODCOMP software implemented to support the bus system. The information presented here, together with a listing of the actual code, is necessary for maintenance or modification of the software.

2.1 The MODCOMP Symbiont

The TMS operations under the MODCOMP-supplied MAX IV (Modular Applications eXecutive) operating system. MAX IV supports both actual I/O devices (referenced directly through I/O requests to the device handler) and imaginary devices (referenced through special system tasks, called symbionts). Symbionts present an imaginary device interface to the programmer while insulating the user from the peculiar aspects of the protocol for the real device.

The MODCOMP interface to the bus is through a standard hardware controller, the Model 4805 General Purpose Data Terminal board (described in [3]). The hardware interface is under the direct control of the MODCOMP CL.HAN I/O handler. For the TMS, a symbiont, named BUSSYM, was chosen to control the handler's operation because the MODCOMP/BIU protocol is different from other MODCOMP device interfaces and because the interface requires that a number of complex functions be performed. Among these are logical multiplexing of communication with multiple bus devices and blocking and deblocking of messages. Use of a symbiont permits the actual MODCOMP/BIU interface to be transparent to the application programmer.

To understand the symbiont's interfaces with MAX IV, some background is needed on the operating system and its I/O logic. Operating system structures used for I/O are discussed in paragraph 2.1.1 of this document. The symbiont's interfaces with a user task are described in paragraph 2.1.2, while the symbiont's

other relationships to the operating system are discussed in paragraph 2.1.3 Paragraph 2.1.4 discusses the host/BIU protocol and source code structure.

2.1.1 MAX IV I/O Data Structures Used by the Symbiont

There are several data structures in the operating system with which the symbiont must deal:

1. I/O Nodes
2. The symbiont's Physical Device Table (PDT)
3. Logical Device Tables (LDTs)
4. Task Control Blocks (TCBs)
5. User File Tables (UFTs)

The significance of each of these structures to the bus symbiont is discussed below, and an example of how the structures are related is given in paragraph 2.1.1.6. The definitions of fields within these structures are given in [8]. In general, not all fields are important to BUSSYM; only those of interest will be mentioned in the following discussion. In any of these structures, a field name is 6 characters long; the first three characters always indicate in which structure the field occurs. For example, NODBUF occurs in an I/O node.

2.1.1.1 I/O Nodes. Every outstanding I/O request is associated with exactly one I/O node. This node contains pointers to the buffer, to the issuing task's TCB, to the LDT for the logical device the task is addressing, and to the PDT for the controller (or symbiont) to which the logical device is assigned. The node also contains fields specifying the size of the buffer, the type of operation requested (read, write, etc.), the options requested, and some workspace in which the operation's progress may be logged. There are also several other fields not of direct interest to the symbiont.

After MAX IV creates the I/O node for a particular I/O request, the operating system links the node into a chain of nodes queued to either a physical device controller or to a symbiont, depending on the device being addressed (see paragraph 2.1.1.3). The node contains pointers to its

predecessor and to its successor in the chain. Ordinarily, the chain is kept in priority order, with nodes from higher priority tasks higher in the chain.

As is discussed below, each physical device and each symbiont have a PDT, which serves, among other things, as the head of the chain of nodes queued to that device (or symbiont). This means that the PDT contains a pointer "down" to the first node in the chain (called the "current" node) and the current node contains a pointer "up" to the PDT.

A symbiont gets all information about users' requests for its services from its own chain. The symbiont is activated by the operating system when the first node appears in its queue, and the symbiont signals completion of an operation in part by removing the node from its queue. The bus symbiont may interleave processing of several nodes, especially when they involve blocking of data from, or deblocking of data to, the data bus. In order to keep track of its progress with the node, the symbiont uses a workspace in the node (field NODACT) which in other devices is ordinarily used for record-skipping operations.

The maintenance of the node chain is not trivial, since any task may queue an operation to the symbiont at any time. Queuing of operations is discussed in paragraph 2.1.3.

2.1.1.2 The Bus Symbiont's PDT. Every device controller and symbiont present in the system has a Physical Device Table. These tables are reserved at system generation (sysgen) time and are linked together into lists by the system. The PDT is the head element of the chain of I/O nodes queued to the device (or symbiont) associated with the PDT. The features of interest in the bus symbiont's PDT are the pointer to the first node queued to the symbiont (PDTFNO) and a bit, called the "shutdown" bit, in the device status word. The shutdown bit is used in part of the coordination among the various routines which may change the node queue. The coordination is discussed in detail in paragraph 2.1.3.

2.1.1.3 Logical Device Tables. Every I/O device -- whether actual or imaginary (supported by a symbiont) -- has a Logical Device Table (LDT), created at sysgen time. Only the devices present in the system's chain of LDT's can be referenced by a program. When an I/O access to a logical device is attempted, the MAX IV Basic I/O System (BIOS) searches the LDTs to determine to which actual controller or symbiont the I/O request should be queued.

In the TMS sysgen, logical devices NO0 through NOC and NI0 through NIC are defined to be supported by the bus symbiont BUSSYM, and each of these devices has an LDT in the system. These device sequences correspond to the "data" and "boot" addresses, respectively (see paragraph 3.1.4.2), of Bus Interface Units on the network. The bus symbiont uses these LDTs to translate the logical device named in an I/O node to a physical device name (LDTNAM), and then uses its own internal table to translate the physical device name to its corresponding address on the bus communications network.

BUSSYM itself accesses the actual hardware interface to the MODCOMP BIU (see paragraph 2.1.4.1) through the BIOS using logical devices BIN (for input from the BIU) and BOU (for output to the BIU). BIN and BOU also each have LDTs in the system, of course.

2.1.1.4 Task Control Blocks. In MODCOMP literature, the fundamental entity of control flow is termed a "task" (this word is used in the same sense that some authors use the word "process"). Whenever the MODCOMP is not processing an interrupt, which it performs using special software in a special hardware state, it is executing some task (possibly the "idle" task). Tasks are usually started ("activated") by the system operator, but they can be started by other tasks. A symbiont task is started by the operating system when the first node is queued to the symbiont by being placed on the chain headed by the symbiont's PDT.

Each task which is present in the system or which is queued to enter the system has a Task Control Block (TCB). These blocks are chained

together in order of priority. After any interrupt, a system routine ("taskmaster") scans the TCB chain for the highest priority task which is ready to execute. The taskmaster then gives that task all necessary system resources, restores the registers and condition codes to the state when the task was last interrupted, and gives the task control. The task keeps control until the next interrupt (which it may issue itself).

Among the resources necessary for a task to execute are map registers. (See [9] for a detailed discussion of the function of these registers.) The MODCOMP IV/35 has 8 map registers, each of which may be allocated to only one task. The registers permit a task to operate with a contiguous logical address space by translating the task's virtual memory addresses into actual main memory addresses. This technique also reduces fragmentation of system memory.

Each map register is actually a table of 256 entries, in which each entry contains the actual memory page number of one of the 256-word virtual pages used by a task. A program uses only one map register at a time, and map register selection is determined by the task's Program Status Word (PSW). A consequence of this arrangement is that the area addressable at one time by a task is limited to 64K words. Furthermore, on the MODCOMP, while there are virtual addresses, the size of virtual memory is limited to the size of real memory since there is no page swapping storage device.

One of the map registers (map zero) is reserved for the operating system; virtual memory addressed by map zero contains all system data structures (including I/O Nodes, PDTs, TCBs, and LDTs). UFTs are considered user data structures and reside in a task's addressing space. Because the bus symbiont is a privileged task, it may address map zero as well as its own map, though a certain amount of overhead is involved in map switching.

Since there are only 8 map registers, there may be more tasks present in the system than there are map registers. In this case, MAX IV selects a task for temporary suspension, copies its map register into a save

area accessible through map zero, and reallocates the map register to another task as needed. The saved map register contents are referred to as a map zero image in the following discussion. The first task's map zero image will be restored to a map register (possibly a different one from the one previously used by the task) before the next time the task is allowed to execute. This operating system activity requires that a privileged task, such as BUSSYM, must inhibit system reallocation of its home map register when it references different address spaces (map zero and user task maps) so that no problems occur when it again uses its home address spece.

The TCB contains the name of the home address map register of a task, as well as the map zero address of the contents of the task's map registers when those registers have been temporarily deallocated, as discussed above. The bus symbiont may be called to serve tasks which no longer have map registers allocated. In such a case task memory must be referenced through the map zero images, rather than through the actual map registers. This type of referencing is supported by the load-and-store via map-image instructions (slower than usual memory-referencing instructions).

The TCB also contains a task status word used by the taskmaster to determine whether the task is waiting for completion of an I/O operation. The bus symbiont must signal the completion of wait-mode operations by resetting a bit in this status word of the calling task.

2.1.1.5 User File Tables. A user task wishing to request an I/O operation from MAX IV first constructs a User File Table (UFT) and then calls the operating system with the address of the UFT. The UFT contains all the information that the system needs to perform an I/O operation -- logical device name, command, buffer addresses, options, etc. (In FORTRAN programs the construction of the UFT is hidden from the programmer by the compiler.) One UFT is required for each concurrent I/O operation for each file.

After the call to MAX IV, the operating system sets a bit in the status word of the UFT, creates, initializes, and queues an I/O node, and does not further alter the UFT until the operation has terminated (successfully or not). At that time, the system resets the bit it set earlier (the UFT busy, or UB, bit) and sets other status bits and the transfer count, as appropriate. If the I/O operation is performed by a symbiont, the symbiont must set the status bits and count. A more detailed description of UFT fields and uses is given in [10].

Since BUSSYM queues operations to physical devices, it too contains UFTs. These UFTs have the same structure as any other task's UFTs.

2.1.1.6 Example of I/O Data Structure Usage. Figure 2.1.1.6-1 illustrates how the MAX IV I/O data structures discussed in the preceding paragraphs relate to each other. For clarity, some of the operating system pointers linking such elements as TCBs and LDTs into chains have not been included. On the diagram, ovals represent hardware components, cross-hatched rectangles represent code, and unhatched rectangles denote data structures. Solid interconnecting arrows show data flow, while dashed arrows indicate pointers, or logical links, among data structures. Control over hardware is shown by a dotted arrow.

The figure shows the linking which occurs when a user task A requests an I/O operation from the BIOS and specifies a logical device which was defined at sysgen time to be a logical device connected to the bus. MAX IV searches its chain of LDTs and locates the LDT for the requested logical device. The LDT indicates that the bus symbiont is to be invoked to handle I/O to that logical device, so the operating system builds an I/O node and links the node into BUSSYM's node chain (which begins with BUSSYM's PDT) in priority order. The node also contains pointers to the requesting task's TCB and UFT for this I/O.

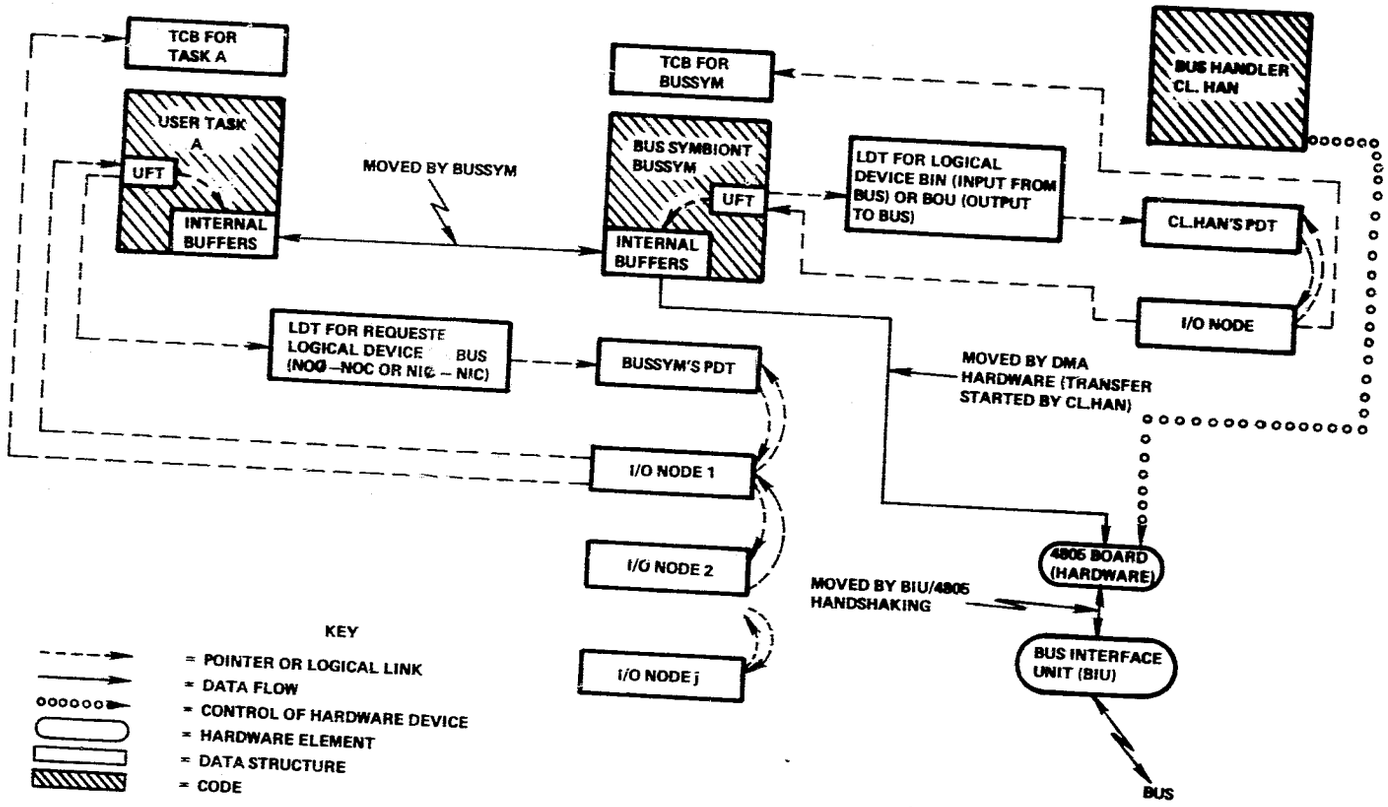


Figure 2.1.1.6-1

MODCOMP Operating System Data Structure Used in Bus I/O

When BUSSYM services the node, it buffers all data through its own initial buffers so that data packet headers (see paragraph 2.1.4.3) can be handled. For output requests, BUSSYM consults an internal table to determine the actual network address for the logical device and prefixes a message header which will ultimately be employed by the BIU, but which appears only as data bytes to the MODCOMP operating system. BUSSYM then requests an I/O either to logical device BIN (for reads from the bus) or BOU (for writes to the bus) and invokes BIOS to carry out the request.

The BIOS treats the symbiont's request with the same procedures as followed for the user task's request. A search of the system's LDT chain shows that BIN (or BOU) is associated with the 4805 handler CL.HAN, and an I/O node is therefore constructed and queued to CL.HAN's PDT. When the handler services the BUSSYM request, the handler issues physical commands to the 4805 board to transfer data to (or from) the symbiont's buffer from (or to) the BIU.

2.1.2 User Communication with the Bus Symbiont

The following paragraphs describe first the procedures by which a user task communicates with BUSSYM and then discuss briefly some of the options supported by the bus symbiont.

2.1.2.1 Procedures for Communication with BUSSYM. The application task directly communicates with the bus symbiont only through the task's UFT (BUSSYM references the task's buffer area, of course, using the map imaging technique discussed in paragraph 2.1.1.4). The task indirectly queues an operation to the symbiont by issuing a write to one of BUSSYM's imaginary devices.

While the operation is pending, the UB (unit busy) bit is set in the application task's UPT status word. The task may check this bit periodically if the operation was issued in quick-return mode; if the operation was issued in wait mode, the task is suspended until the operation is done. No other bits in the status word are changed until the completion of the operation.

It should be noted that the bus symbiont places no time limit on how long a request may wait for service. If a task needs to limit the time to wait for a read, for example, it must issue the read in quick-return mode and use MAX IV timer services. The task may, of course, terminate any of its outstanding I/O operations at any time for any reason.

If the operation is successful, the status word is reset to zero, and the count of bytes transferred is placed in the UFT field UFTBCT.

If the user tries to specify a buffer which falls outside his allocated memory, the status word is returned with bits 0 and 8 set (bit 0 is the leftmost, or most significant, bit). If the UFT option word for the operation specifies system recovery for errors, the application task is aborted and the following message displayed on the operator console:

!ABORT (BUS XLM nnnn)/taskname/overlayname

where "taskname" and "overlayname" identify the user task and "nnnn" gives a task address near where the offending operation was issued.

If the application task references an imaginary device DDD which was not installed in MAX IV at sysgen time, the operating system aborts the task and shows an error code of IFN DDD or ASG DDD (depending on the type of reference) in a message of the form shown above. If, on the other hand, the imaginary device was specified at sysgen but has not been inserted in the symbiont's internal table of device names, BUSSYM sets bits 0 and 3 in the UFT status word, and, if the system recovery bit is set in the UFT option word, the task is aborted and an error message showing a cause of BUS DEV (format as shown above) is given.

2.1.2.2 Request Options. Most of the capabilities usual to communications devices are provided by BUSSYM and can be requested through UFT options.

For example, both binary and ASCII modes of I/O are supported. When the BI (binary) bit is set in the UFT option word at the time of request, completion of the operation is determined by the buffer size specified either in the UFT or in registers at the time of the I/O operation request. Reads and writes proceed until the user buffer is filled or emptied, respectively, unless an abnormal termination is encountered. Special characters are forwarded by the symbiont without any action being taken on them.

When the binary bit is not set, however, the usual MODCOMP ASCII rules are followed. This means that for a write, the operation proceeds only until a word containing binary zero is written; for a read, the operation is ended on the receipt of binary zero.

The symbiont does not support data chaining or use of a buffer in any map other than the calling task's.

2.1.3 Multiprogramming Considerations

In a multiprogramming system, such as MAX IV, many tasks may be taking turns using a single processor. A task proceeds until it is interrupted, or until it must wait for some I/O operation to complete. At that time, control of the processor is given to other tasks until the first task is again ready to run and no higher-priority tasks are ready.

Ordinary tasks are independent of each other and have no need for intertask coordination beyond what the operating system provides in the allocation of system resources. When a user task is interrupted and later regains control, the interruption is transparent to the task because its state is saved at the time of the preemption and restored before control is returned.

A symbiont, however, faces a more complex situation. First, the queue of I/O nodes with which it works is accessed by other routines asynchronously. The symbiont, for example, may be interrupted by the system so that a node can be added to (or deleted from) the node queue.

The node queue may thus change while BUSSYM is working with it. Second, the buffers used to exchange data with a calling task are also subject to outside control. If, during the course of an interrupt (during which the symbiont is preempted), the calling task is killed and its memory deallocated, the symbiont may attempt to access memory which has been deallocated. In that case the symbiont is aborted by the operating system.

These two situations are representative of the general problems of insuring that the node queue is correct when it is used and insuring that memory is not deallocated while the symbiont is accessing it. The node queue problem is mitigated by use of the shutdown bit in the PDT, which is at the head of the node queue. Whenever another operating system routine alters the queue in any way, it resets the shutdown bit before releasing control. In order to insure that a node is truly present in the queue, BUSSYM therefore first sets the shutdown bit, then follows the chain to the node, and finally tests the shutdown bit again. If the bit is still set, then the node is still in the queue. If the bit has been reset, however, the node may have been removed. Any time BUSSYM finds the shutdown bit reset, it restarts its search of the queue from the top after setting the bit again, and continues until it finishes its search without being interrupted.

The problem of memory deallocation is solved by a second feature of MAX IV: a privileged task may lock out the taskmaster. Any interrupts occurring while the taskmaster is locked out are handled normally, but during the lockout time, the taskmaster does not scan the list of tasks to decide which one next receives control. Instead, control always returns to the task which was interrupted. If, during the interrupt processing, the interrupt routine decides to abort a task (for a reason such as the occurrence of an I/O error), it does this by setting a bit in the victim's TCB. The interrupt routine itself does not deallocate the aborting task's resources: these resources remain allocated until the taskmaster is again enabled, at which time it reviews the task queue, deallocates any tasks with the TCB abort bit set, and gives control to the highest priority ready task.

The symbiont must lock out the taskmaster whenever BUSSYM enters map zero addressing space so that the symbiont's own map registers are not deallocated (see paragraph 2.1.1.4). In addition to this lockout, to avoid having any user task's buffer deallocated while BUSSYM is interrupted, the bus symbiont does not release the taskmaster until either the bus symbiont is finished with the user task's buffer or the bus symbiont is willing to check that the node is still present in the queue before addressing the buffer again. The node's continued presence in the queue implies the buffer's continued allocation in memory, since the node is removed before memory space is deallocated.

These problems combine to force the symbiont into the following sequence of steps when it wants to address a user task's buffer:

1. Lock out taskmaster
2. Select map zero operand addressing
3. Set shutdown bit in PDT
4. Find I/O node in the queue
5. If shutdown bit is reset, return to Step 3
6. Move data to/from user's buffer or UFT
7. Select symbiont's own map
8. Release the taskmaster

The bus symbiont consequently has the taskmaster locked out for a large precentage of the bus symbiont's execution time. If the system has many tasks at higher priorities than BUSSYM, this taskmaster lockout may degrade system performance by letting the bus symbiont keep control for relatively long periods while higher priority tasks are ready to run. In the TMS, however, the bus symbiont is the highest priority task which needs to do any significant amount of work.

2.1.4 MODCOMP/BIU Protocol

As described in [1] and [4], a Bus Interface Unit (BIU) is a microprocessor-based high-speed modem which communicates simultaneously with other BIUs (via the data bus) and with its subscribers (via the subscriber interface). The method of inter-BIU communication is discussed in [4] and [6] and will not be further considered here. The MODCOMP's BIU communicates with the MODCOMP using a Direct Memory Access (DMA) interface, the hardware of which is described in [3] and [11]. The following discussion gives design and implementation details for the protocol.

Paragraph 2.1.4.1 gives background on the DMA interface hardware, as it influences the software. Paragraph 2.1.4.2 discusses the requirements levied upon the protocol used in BIU/MODCOMP communications. Paragraphs 2.1.4.3 through 2.1.4.10 describe the detailed design of the protocol from several perspectives. A bus symbiont overview flow diagram appears in paragraph 2.1.4.10.

2.1.4.1 The MODCOMP/BIU DMA Interface. Interaction between the MODCOMP and its BIU occurs via a standard MODCOMP 4805 DMA interface board (see [3] and [11]). In each direction, data are passed in 16-bit parallel mode. The interface is fully handshaken so that neither the MODCOMP nor the BIU ever passes a data word unless the other party is ready. This handshaking is transparent to the communications software and is consequently not further mentioned here.

In the MODCOMP the standard computer link I/O handler CL.HAN is used to control the interface (see Figure 2.1.1.6-1). Its purpose is to mediate between the 4805 hardware and the operating system by issuing commands to the 4805 and interrogating it. Unlike the symbiont, the handler is an interrupt routine. It is invoked whenever the symbiont issues an operation to the BIU, whenever the BIU interrupts the MODCOMP, and whenever the 4805 completes an operation to the BIU. The handler will not be discussed further in this paper, since it is a standard MODCOMP software product and is transparent to the symbiont.

The 4805 can pass data in only one direction at a time -- it is a half-duplex rather than a full-duplex device. Therefore, to reverse the direction of data flow, or "turn the line around," whatever operation is using the 4805 must be ended, and a new operation started in the opposite direction. The protocol signals described in paragraph 2.1.4.4 are used to coordinate this process.

The BIU includes a status register which the 4805 interrogates when an interrupt occurs on the MODCOMP/BIU interface. (The interrupt may be triggered by the BIU or internally by the MODCOMP.) The MODCOMP handler then takes a "snapshot" of the BIU status (passed through from the 4805), merges it with some internal status bits, and returns the result to the bus symbiont. If the operation is "terminated" (aborted) by the symbiont, however, no interrupt is issued, and the BIU status is not interrogated. The symbiont gets no status information about operations that it aborts.

The BIU uses two bits in its status register to signal the symbiont (see paragraph 2.1.4.5). These are the "timeout" (referred to as S4 in hardware discussions such as in [3], but termed TO for clarity in this paper and in [6]) and "more data" (S5 in hardware references, MD in this paper, and INRDY in [6]) bits. Like any other status bits, they are available to the symbiont only after the handler has read them from the 4805 hardware lines. This only happens after an interrupt, so changes made by the BIU to its status register are not seen by the symbiont until the next interrupt has occurred on the MODCOMP/BIU interface.

The 4805 also has some status lines, which the BIU may interrogate. One of these, the "busy" line (BUSYN), tells the BIU whether the 4805 is working on an operation or not. That the 4805 is working does not necessarily mean that data are being transferred, since both the BIU and the 4805 must cooperate to transfer data. It does mean that the handler associates some operation request with the 4805, and that the 4805 is ready, or very shortly will be ready, to send or receive data.

When the BUSYN line is asserted, the BIU may end the operation by issuing an interrupt (accomplished by pulsing the 4805 EXTSIN interface line). If the BIU interrupts while BUSYN indicates that no operation is running, the handler detects the interrupt as an error and writes an error message to the system console. If the BIU interrupts a write operation from the MODCOMP, the handler and operating system consider that a serious I/O error has occurred.

Because of these handler reactions to BIU interrupts, in the NASA TMS MODCOMP, an artificial operation (termed a "dummy read" because there is no demand for data transfer) is issued to the BIU by the symbiont whenever there are no other operations to be done. This action causes BUSYN to be true and also provides an operation whose interruption does not cause the handler to take error action. The BIU can then obtain the symbiont's attention by interrupting the dummy read, or the symbiont can terminate the dummy read and issue a write if the need arises.

Dummy reads are like "real" reads except that (1) the symbiont may terminate them at any time, (2) the symbiont expects no data from them, and (3) while they are outstanding, the symbiont periodically searches the node queue for writes. When a write request appears, the read is terminated by the symbiont before the write is issued. Only the symbiont knows whether a read is a "dummy read" or a real read; it keeps track of the difference by making note of the circumstances under which the I/O was issued.

The portions of the interface operation dealing with dummy reads are discussed more fully in paragraph 2.1.4.5.

2.1.4.2 Design Requirements. The main requirement upon the MODCOMP/BIU protocol is that it permit either participant to limit the flow of data. The BIU has finite internal buffers and may not be able to pass data into the cable bus as fast as it is getting data from the MODCOMP.

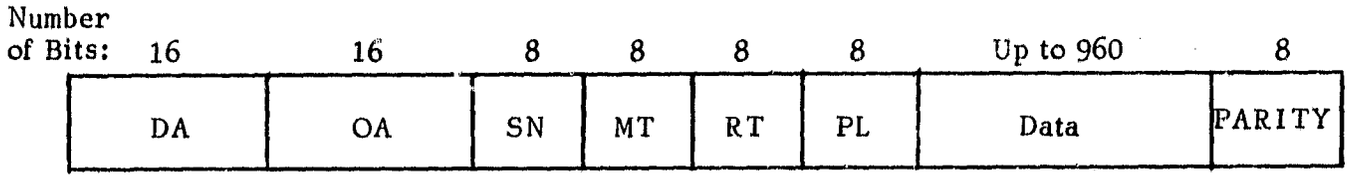
Similarly, the symbiont may not be able to process its input immediately, since it operates in a multiprogramming environment. Therefore, the symbiont and the BIU both need to be able to limit the data rate.

The data transfer must be fast in both directions, but especially on output from the MODCOMP, because for the TMS application, the volume of data output is much greater than input. Because of the large data volume, the processing overhead in the MODCOMP must be kept small.

The issue of error control need not be addressed in the MODCOMP/BIU protocol because the short physical connection between the BIU and the 4805 is expected to be relatively error-free, and the additional work of checking error-detecting codes would be a significant burden on the MODCOMP. (The BIU uses error-detecting codes in its conversations with other BIUs; see [4]). Any process requiring very low error rates may, of course, implement error-checking with a higher-level protocol in its own code.

2.1.4.3 Packet Formats. Messages (or "packets") are passed between the BIU and the MODCOMP in almost exactly the same form as they are passed between BIUs on the data bus. The formats of all messages on the bus are described in [4]; the format of messages passed by the MODCOMP BIU to the symbiont is reviewed below.

Input packets may be up to 128 bytes long (64 MODCOMP 16-bit words). They consist of an eight-byte header, followed by up to 60 words of data. On the network, the packet is followed by one byte of error-detection code. This last byte is the only difference between packets as they appear on the cable network and input packets as they are passed to the MODCOMP. Figure 2.1.4.3-1 illustrates the packet format, which is further discussed below.



DA = Destination Address
OA = Origin Address
SN = Sequence Number of Packet
MT = Message Type
RT = Retry Count
PL = Packet Length
PARITY = Longitudinal Parity Byte

Figure 2.1.4.3-1 Packet Format

The first two bytes in the packet (bytes 0 and 1) are the Destination Address (DA), which is the network address of the device to which the packet was sent. All packets with a DA of 0000 are given to the symbiont by the BIU. In the TMS, the only packets with this DA are network status messages (see paragraph 2.2). Data packets (indicated by a special value in the MT field, which is discussed below) which are addressed to DA 0200 are also passed to the symbiont.

Bytes 2 and 3 of the packet are the Origin Address (OA). This is the network address of the device which created the packet. The MODCOMP accepts packets from any origin on the network.

Byte 4 of the packet is the Sequence Number (SN). Successive packets transmitted by any BIU are given sequence numbers in the range hexadecimal 00 through FF, so that receiving BIUs can detect duplicate packets.

Byte 5 of the packet is the Message Type (MT). Several types of messages are transmitted on the TMS bus system. Only two types of messages, however, are transmitted to the MODCOMP from its BIU -- status messages (coded as hexadecimal DB) and data messages (coded as hexadecimal 02). The symbiont takes the data messages from the BIU and passes them on to user tasks to satisfy outstanding I/O requests. Status messages, however, are processed by the symbiont and are not seen by applications software (see paragraph 2.2).

Byte 6 is the retry count (RT), which indicates how many times this packet has been retried (because of a missing acknowledgement) on the network.

Byte 7 of the packet (and the last byte in the header) is the Packet Length (PL) field. This field contains a number which is one less than the byte length of the packet, where the length includes the header, but not the error-checking byte. In input packets, this value never exceeds 127. Odd byte counts may occur. In this case, the MODCOMP BIU pads the input packet with a trailing byte of unspecified bits, to be able to transfer a full 16-bit word to the 4805. The BIU does not alter the PL field on input packets, however, so the symbiont may check the PL to see whether the last byte is data or filler.

The rest of the input packet (after the header) is data, for data-type messages, or a variety of status fields, for status messages. BUSSYM always strips off the packet header before passing data to a requesting program.

Output packets from the MODCOMP symbiont are the same as input packets, except for the following differences:

1. Output packets may be up to 1024 bytes long, including the header, which is added by BUSSYM. The length is indicated in the RT and PL fields (considered together as one 16-bit unsigned integer).

2. The OA field in output messages is present but ignored by the BIU. The BIU inserts the proper OA before passing the packet to the network.

Permitting the bus symbiont to send a large packet to the BIU (which then reformats the large packet into several small packets before transmitting the information on the network) reduces the MODCOMP's processing overhead, since MAX IV requires about 1 millisecond to process a write request.

Transfers between the BIU and the user task are fully buffered within the symbiont. Consequently, if a user task requests a read of more than 128 bytes (one network packet), the symbiont actually issues multiple reads to the BIU to obtain the requested amount of data. Similarly, if a user task issues a write of more than 1024 bytes, the symbiont breaks the write into several pieces, each of which is performed separately.

2.1.4.4 Protocol Signals. The word-by-word transfer of data between the BIU and the 4805 uses the 4805's standard handshaking lines and logic, in which the recipient signals each time it is ready to accept another word of data. The lines are described in [3] and [11], and the use of the lines by the BIU is discussed in [6], so further discussion of the word-by-word handshaking is not included here. BUSSYM does not have access to these handshaking lines.

The "more data" (MD) bit is set by the BIU if and only if the BIU has an input packet ready for the MODCOMP, in addition to any packet the MODCOMP may already be reading.

The "timeout" bit (TO) is set by the BIU whenever an operation seems to be taking too long. The purpose of the BIU's interrupting in this case is to ascertain that the MODCOMP has not crashed. Timeout values and lengths of reads and writes have been chosen so that under normal conditions, I/O operations for transfers of data (not "dummy reads") should not be interrupted because of the occurrence of this condition. During an idle period, however, dummy reads are routinely interrupted by the BIU at regular intervals. The interrupts may be interpreted as the BIU's saying "I'm okay; are you okay?".

The BIU uses the 4805's "busy" line BUSYN to find out whether the last operation is still active, i.e., whether the symbiont or a system failure has ended the operation. The line is set indirectly by the symbiont, whenever it issues an operation. It is reset automatically when the operation ends (either normally or abnormally).

2.1.4.5 Protocol Events. The normal idle state of the MODCOMP/BIU interface is for the MODCOMP to have a dummy read outstanding to the BIU. As mentioned above, this provides an interruptible operation which can be abnormally terminated by the MODCOMP if a user task generates a write request, or which can be terminated by an interrupt from the BIU if a packet arrives from the network for the MODCOMP. Occurrences which disturb the idle state or which cause transitions between various active states are termed "events."

An "event" significant to the MODCOMP/BIU protocol occurs when one of the following three things happens:

1. the symbiont issues an operation to the BIU,
2. the BIU interrupts the MODCOMP while no operation is underway between the BIU and the MODCOMP, or

3. some operation to the BIU is ended in one of the following ways:

   (a) terminated by the bus symbiont
   (b) interrupted by the BIU
   (c) completed by the transfer of the requested number of words
   (d) aborted by a timeout in the handler

The first class of events requires no comment. Operations issued to the BIU are treated like any other I/O work by the operating system. The BIU discovers when an operation has been issued to it by polling BUSYN (the 4805 "busy" line), and it determines whether the operation is a read or a write by testing the 4805's handshaking lines.

Events in the second class occur only when the MODCOMP has suffered some failure, causing it not to issue any operation (even a dummy read) to the BIU for about 0.1 second.

The third class of events consists of the four possible ways in which an operation may be ended. The first way is that the symbiont may "terminate" (abort) the operation. This is normally done to end a dummy read when the symbiont discovers that some user program has requested a write operation, but may also be done at the request of the user task which issued the I/O.

The second way an operation may be ended is by an interrupt from the BIU. When this interrupt is issued, the BIU status register contents are recorded and passed to the symbiont, and the symbiont is signalled that the operation is complete. This is the usual way of ending reads by the MODCOMP from the BIU, but this interrupt can also be generated if an operation takes too long. Presently, the BIU interrupts the MODCOMP if the transfer of an incoming packet (up to 128 bytes) to the MODCOMP takes more than 0.02 seconds or if the transfer of an outgoing packet (up to 1024 bytes) takes more than 0.05 seconds. A dummy read (issued when neither the BIU

nor the MODCOMP has traffic for each other) is terminated after 0.1 seconds if it is not ended earlier by the arrival of traffic.

The third way an operation may be ended is by reaching the requested byte count. When this happens, the 4805 interrupts the MODCOMP, the BIU's status is recorded and passed to the symbiont, and the symbiont is signalled that the operation (usually a write) is complete.

The fourth and final way in which an operation may be ended is when a timer expires in the MODCOMP device handler. In the sysgen of MAX IV for TMS, the bus is specified as an untimed device, but because of operating system bugs, the device is considered timed nonetheless. The time interval and one status bit meaning are set in MAX IV Level C using the following sysgen patches:

```
ORG TCTCL
DFC #8000      (Patch status bit)
ORG PDTCL-10
DFC 5000       (Set timer value)
```

Any time an operation lasts longer than about 50 seconds the handler aborts the operation and signals an error to the symbiont. The BIU is always supposed to interrupt operations before this point, so the handler's abort of the operation always means that the BIU has failed.

2.1.4.6 State Diagram of MODCOMP Interface Protocol. Figure 2.1.4.6-1 represents the states of the MODCOMP/BIU protocol, and the legal transitions between those states. Each transition is numbered, and the key given in Table 2.1.4.6-I shows in decision table format the conditions which cause the transition and the ensuing results. The transitions are described with principal emphasis on how they affect the MODCOMP; details on BIU behavior in the corresponding situations are found in [6].

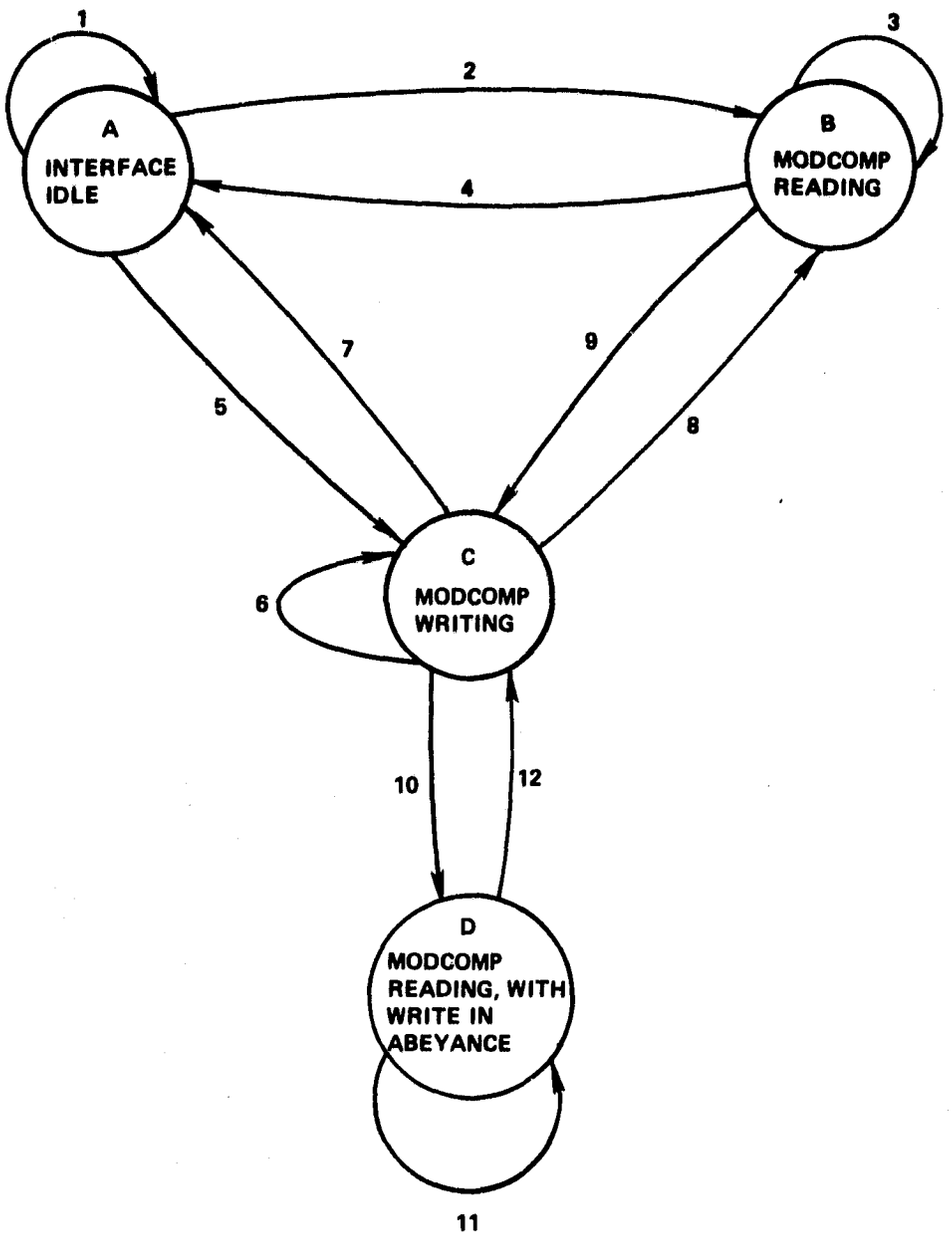


Figure 2.1.4.6-1

MODCOMP/BIU Protocol State Diagram

Table 2.1.4.6-I

Explanation of MODCOMP/BIU Protocol State Diagram Transitions

Transition	1	2	3		4	5	6		7	8	9	10	11	12
Conditions														
a. Reason for BUSSYM activation[2]	B	B	B	B	B	W	D	B	D	D	B	B	B	B
b. MD bit set	N	Y	Y	–	N	–	N	N	N	Y	N	Y	Y	N
c. TO bit set	–	–	N	Y	N	–	N	Y	N	N	N	Y	–	–
d. BUSSYM has a queued write	–	–	–	–	N	–	Y	–	N	–	Y	–	–	–
Actions														
a. BUSSYM issues dummy read	X				X				X					
b. BUSSYM issues read for data[1]		X	X							X		X	X	
c. BUSSYM terminates dummy read						1								
d. BUSSYM issues next write						2	X				X			
e. BUSSYM reissues last write								X						X
f. BUSSYM reissues last read				X										

NOTES:

1. BUSSYM will always issue a read if the BIU offers data. If no read request is queued, the data are discarded
2. If BUSSYM is awakened by a request from a user task and the interface state is not idle (state A) the request is queued and BUSSYM awaits the next interrupt

Key to "reason for BUSSYM activation":

B = Interrupt from BIU

W = Arrival of write request from user task

D = Completion of DMA operation

Whenever an event occurs, the protocol makes some transition. (The events are described in paragraph 2.1.4.5.) Which transition is made depends on which event occurs, what the state of the protocol was before the event, whether the symbiont has write requests in its queue, and whether the MD and TO bits of the BIU status are set. As will be seen in the figure, some transitions do not change the state of the protocol.

State D ("MODCOMP Reading With Write in Abeyance") differs from State B ("MODCOMP Reading"), even though in State B the MODCOMP may also have new writes ready to begin. "Write in Abeyance" means that when the MODCOMP again gets a chance to write, the data transmitted will be a repeat of the last write the BIU interrupted. Since the symbiont's request queue of I/O nodes is ordered by priority instead of being simply First-In-First-Out (FIFO), this differs from pushing the write back into the request queue. The same write must always be retried first because the BIU will discard as many bytes of the retried write as it passed to the network on the first attempt.

2.1.4.7 Symbiont's Rules. A protocol may be defined through the set of rules obeyed by each participant. The rules which govern the bus symbiont are given below. The BIU's rules are given in the next section, and the protocol states and transitions resulting from the participants' following these rules appear in paragraph 2.1.4.6.

1. At MODCOMP system startup, the symbiont issues a read to the BIU and waits for a response, even if the symbiont has writes ready almost immediately. (The symbiont therefore must be an autostart task, which is given control at boot time.)

2. The symbiont always issues a dummy read to the BIU when the symbiont has nothing else to write, unless the BIU has indicated with the MD bit that there is input ready for the MODCOMP. The dummy read is used so that the BIU may signal the symbiont immediately, by interrupting and ending the dummy read, whenever input arrives at the BIU from the network.

Without the dummy read, the handler would not know what to do with an interrupt from the BIU.

3. The symbiont "terminates" (aborts) the dummy read and issues a write if it discovers a write request from a user program before the BIU ends the dummy read.

4. At the end of a write, the symbiont examines first the count of the words actually transferred and then the TO and MD bits from the BIU.

(a) If the number of words transferred is less than the number of words requested, the symbiont reissues the last write at once.

(b) Otherwise, if only the TO bit is set, the symbiont reissues the last write (from its internal buffer shown in Figure 2.1.1.6-1) immediately.

(c) Otherwise, if only the MD bit is set, the symbiont considers the write to have been successful (since the data were accepted by the BIU) and issues a read to the BIU for an input packet.

(d) Otherwise, if both bits are set, the symbiont saves the contents of its internal output buffer for later retransmission, and issues a read to the BIU for data.

(e) Finally, if neither bit is set, the symbiont considers the write to have been successful and either starts another write (if one is available) or enters the idle state by issuing a dummy read to the BIU.

5. After completing any read (dummy or other), the symbiont issues a read for data if and only if the MD bit was set after the first read. The symbiont thus issues reads for data until the BIU indicates by resetting the MD bit that no more input is ready. Reads "for data" are distinguished

from dummy reads in that the symbiont never aborts a read "for data," while it may abort a dummy read at any time, in response to a write request from a user task.

Reads "for data" are for one word more than a network packet to insure that the read is always terminated by an interrupt from the BIU rather than by an interrupt from the DMA word counter's reaching zero.

6. When the bus symbiont finally stops reading for data, because it finds the MD bit reset, it retries any interrupted write before starting a new write, regardless of the priorities of tasks requesting the write.

2.1.4.8 BIU's Rules

1. The BIU interrupts the MODCOMP whenever an operation exceeds the time limits defined in paragraph 2.1.4.5. The purpose of these interrupts is to confirm that the MODCOMP is still properly operating and to indicate that the BIU is also still healthy. When the BIU interrupts according to this rule, the TO bit is always set in the BIU status word. The MD bit is also set if appropriate at the time of the interrupt.

2. When the BIU has data for the MODCOMP, it transmits only after it has signalled the MODCOMP by interrupting with the MD bit set.

3. The BIU ends reads issued to it by interrupting the MODCOMP. It ends dummy reads (i.e., reads issued before the BIU has signaled the MODCOMP that there is input) without passing any data. Other reads (i.e., the symbiont's reads "for data") are ended only after a full network packet has been passed to the MODCOMP (unless Rule 1 requires an interrupt.) No more than one packet (maximum of 128 bytes) is ever passed between interrupts. If Rule 1 requires that the BIU interrupt the MODCOMP in mid-packet, the TO and MD bits are both set, and when the MODCOMP next reads the BIU, the BIU retransmits the same packet from the beginning.

4. When the BIU interrupts the MODCOMP in mid-write, as may happen due to Rule 1 above, it keeps a count of how many bytes from the write have been released to the network. This is to avoid duplicating data on the network when the MODCOMP later reissues the write.

5. The BIU passes the MODCOMP symbiont all data packets addressed to the MODCOMP and all packets addressed to network address 0000.

2.1.4.9 Protocol Operation. In order to further clarify the protocol description, some possible sequences of events are given below. First protocol initialization will be discussed, then a typical write by the MODCOMP, and finally a typical input to the MODCOMP from the BIU.

Normally, when the MODCOMP is booted, the BIU will already be powered up. Whenever it is powered up and idle, the BIU issues interrupts to the MODCOMP at the rate of one every 0.1 seconds, regardless of whether the MODCOMP is "alive." When the bus symbiont is activated at MODCOMP system startup, it issues a read to the BIU. The next time the BIU issues its interrupt, the interrupt is detected and a status word passed to the symbiont.

The symbiont examines the MD ("more data") bit in the status returned from the read. If the bit is set, the symbiont issues another read. If, on the other hand, the "more data" bit is reset, the symbiont examines its queue of requests for services (I/O nodes). If the symbiont finds a write, it issues a write to the BIU. Otherwise, it issues a dummy read to the BIU and continues occasionally scanning its request queue in case a write should appear.

Any write issued by the symbiont continues until one of three things happens. Either the byte count for the write expires, indicating that the message has arrived successfully at the BIU, or else the 0.05-second timer in the BIU runs out before the write is done, or else the write is ended by the MODCOMP without the transfer of all the requested bytes

(this latter condition results from unexplained behavior within MAX IV). At the completion of the write (normally or abnormally), the symbiont follows the rules described in paragraph 2.1.4.7.

The symbiont issues "real" reads (as opposed to "dummy" reads) whenever it detects the "more data" bit set on any status from the BIU. These reads are only terminated by the BIU, or, in the event of BIU failure, by the timeout feature in the MODCOMP handler. Dummy reads, on the other hand, may be terminated either by the symbiont (whenever it discovers a write request in its queue), or by the BIU.

On completion of a read, the symbiont checks the message type of the input packet. If the packet is a status message, the bus symbiont updates its network status records. If the packet is a data message, the symbiont notes the packet's origin address and searches its queue for read requests issued to the corresponding device. If such a request is found, the symbiont moves data from the packet into the requester's buffer; if no request is found, the data is discarded. If the MD bit is set, the symbiont then issues another read. Otherwise, it issues a dummy read or a write, depending on whether any write requests are in the queue.

2.1.4.10 Symbiont Flow Diagram. Figure 2.1.4.10-1 shows an overview flow diagram of the bus symbiont, and, in particular, illustrates how the MD and TO status bits from the BIU are used to control the MODCOMP's responses to the MODCOMP/BIU protocol. An important module of the bus symbiont is the STUFF$ routine, which disposes of incoming data from the network. Figure 2.1.4.10-2 is an overview flow diagram of STUFF$ and shows particularly how the taskmaster lockout and shutdown bit considerations discussed in paragraph 2.1.3 are handled.

The flow diagrams in this paragraph show the logical structure and operation of the bus symbiont, but omit some detail for the sake of clarity. The flow diagrams (and others in this paper) are in the form of Nassi-Shneiderman charts, the structure of which is described in [18].

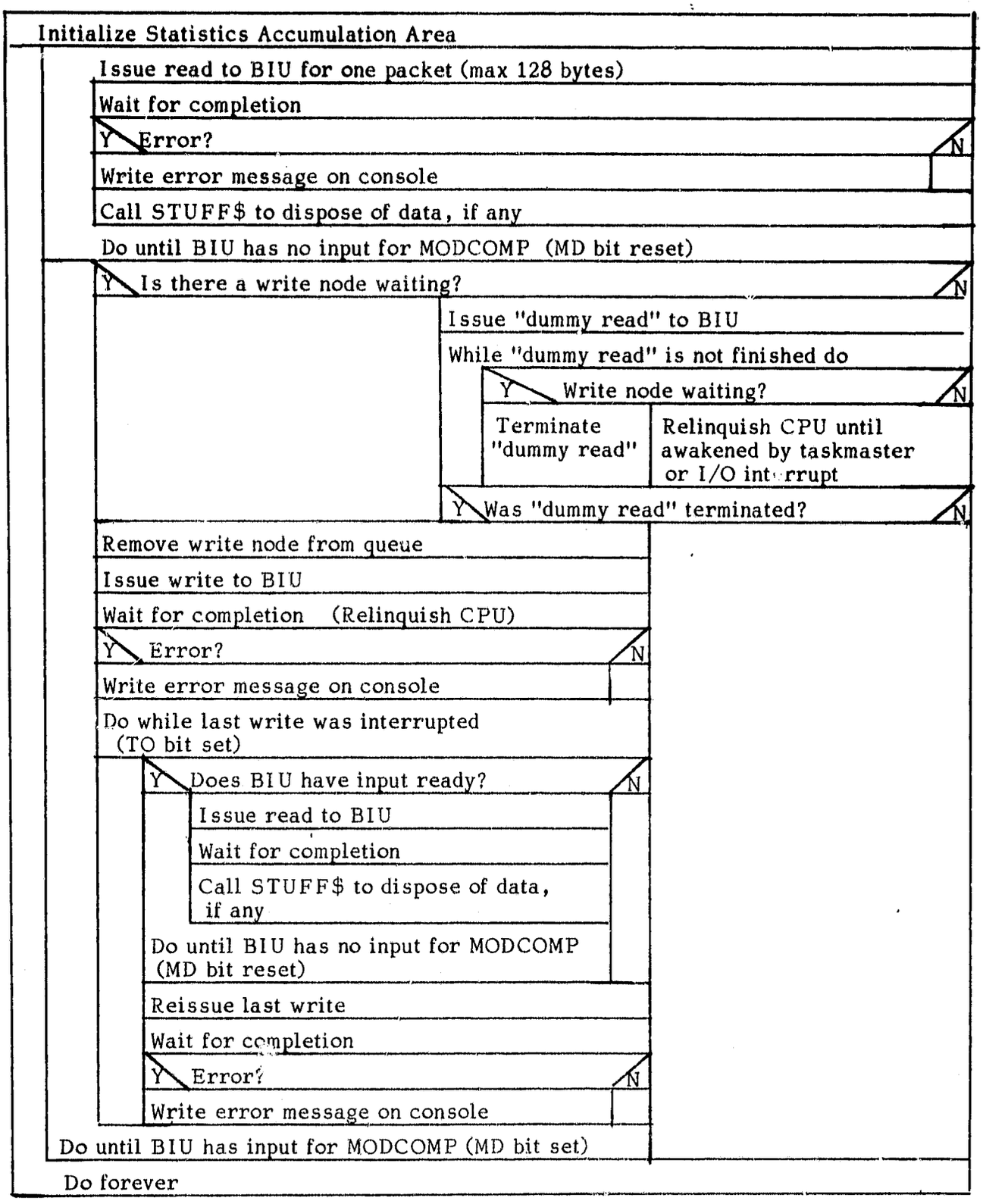


Figure 2.1.4.10-1

Overview Flow Diagram of MODCOMP Bus Symbiont BUSSYM

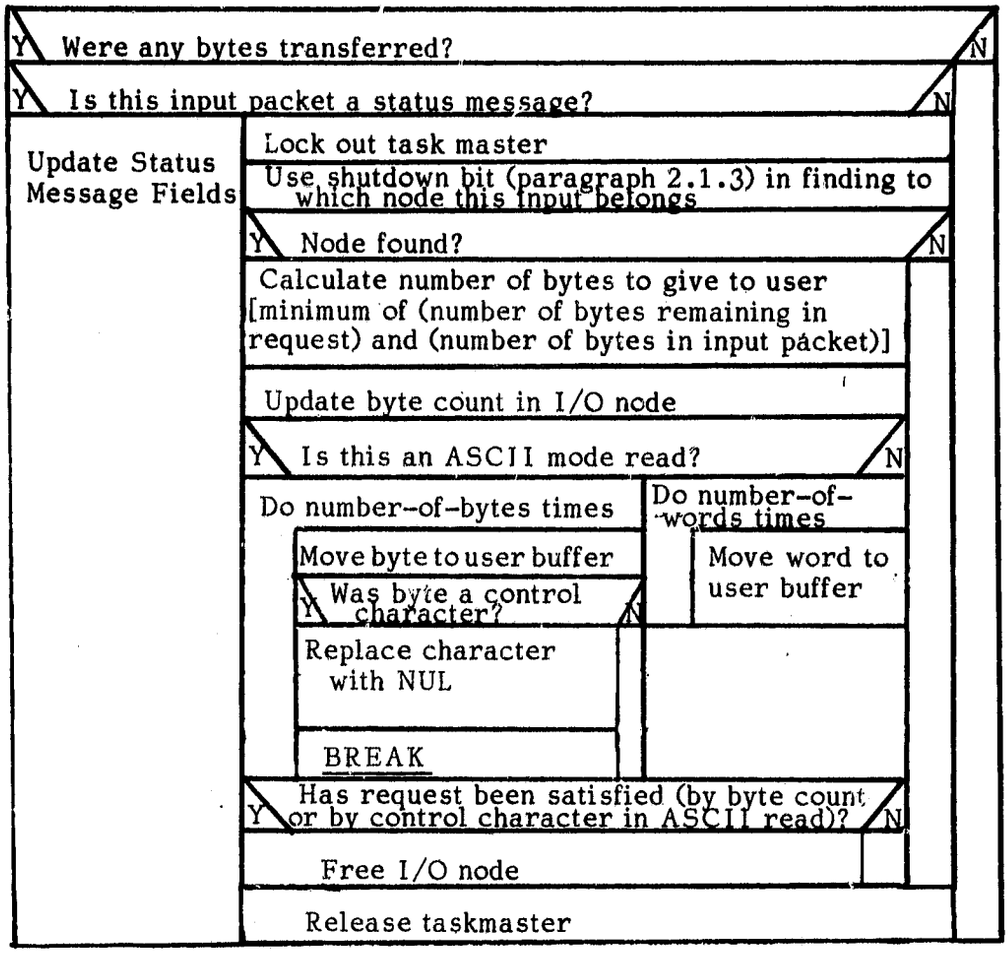


Figure 2.1.4.10-2

Overview Flow Diagram of Bus Symbiont's STUFF$ Routine

All reads for data issued by the symbiont are issued in quick-return mode, so that the symbiont must explicitly wait for an operation to complete, by using the MAX IV RELTILL (relinquish until an event occurs) service, when that waiting is desired. When the symbiont has issued a dummy read, however, it relinquishes the CPU only until the next interrupt, at which time BUSSYM rechecks its node queue and the BIU status. Writes are issued in wait mode, so that the symbiont does not receive control back until completion of the write.

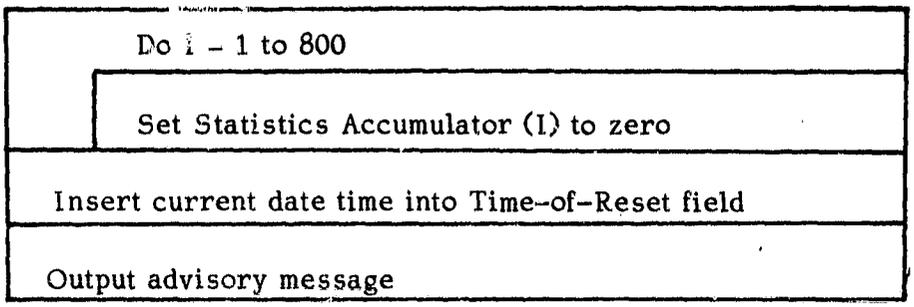
2.2 Bus Statistics Processors

As indicated in paragraph 2.1.4.9, BUSSYM processes incoming status messages itself and maintains a set of counters for each status message field for each BIU active on the bus system. The structure of a status message is described in [4], and the meaning of each field is discussed in [7].

The fields are maintained in an 809-word shareable common area STATAB, in which the first 8 words contain the date and time of the last reset of the statistics counters, the next word contains the number of MODCOMP handler-detected bus errors (roughly equivalent to the number of writes interrupted by the BIU), and the following 800 words contain status data for the terminals on the net. The 800 words provide space for information about 16 active BIUs; for each BIU, 25 words are used to contain the last status message and 25 words are used to maintain cumulative counts of status fields.

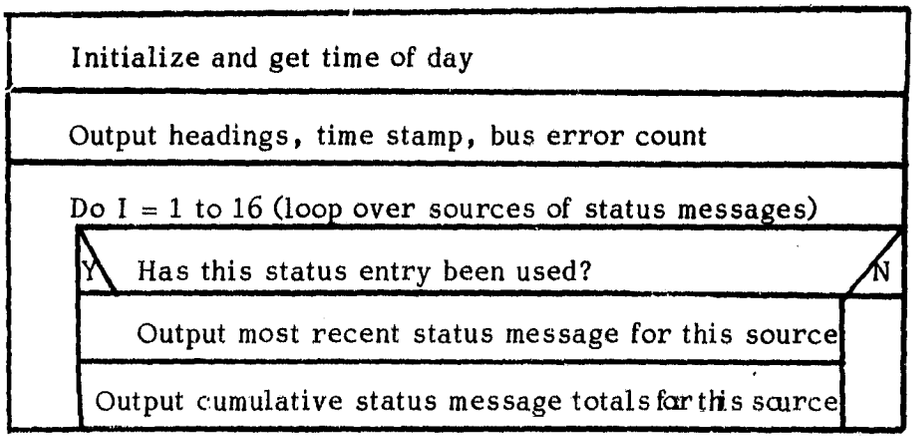
The statistics counters may be reset to zero by the MODCOMP program RESET and may be displayed by the MODCOMP program STATS (called STATS2 when activated from the operator console). Reference [12] contains procedures for using these programs.

Figure 2.2-1 contains a flow diagram of the RESET program, while Figure 2.2-2 illustrates the flow of the STATS program. The versions STATS and STATS2 are identical except for output formatting; STATS is designed for an 80-column-wide CRT, while STATS2 is implemented for a 132-column-wide printer.



NOTE: This routine references statistics accumulators located in a common block accessed by the bus symbiont.

Figure 2.2-1
Statistics Counters RESET Program



NOTE: This routine references statistics accumulators located in a common block accessed by the bus symbiont.

Figure 2.2-2
Statistics Display Program (STATS)

2.3 MODCOMP Programs to Transfer Files between NOVA and MODCOMP Computers

The basic MEGATEK 5000 terminal as used in the TMS contains a NOVA computer with volatile main memory and no mass storage, so that if the unit is powered off, the terminal program is lost. A copy of the terminal program is consequently kept on mass storage at the MODCOMP and downloaded over the communications bus into terminals as needed. Procedures for this bootstrapping of terminals are given in [12].

The development and maintenance of the terminal software, on the other hand, is conducted on a single MEGATEK which is equipped with floppy disk storage, but which is not planned to be a regular part of the TMS. As a result, programs have been developed to transfer the absolute (executable) terminal program from the development MEGATEK to the MODCOMP, and to transfer the program from the MODCOMP's storage to any MEGATEK terminal. These programs all use the bus system for communications.

The following paragraphs describe BOOTSV, which is the MODCOMP program which receives the absolute from the development MEGATEK, and BOOT, which is a MODCOMP program to boot MEGATEKs. The MEGATEK program corresponding to BOOTSV is discussed in paragraph 3.2. One should note that the booting function is included within the TMS monitor [13]; the BOOT program exists primarily to permit booting of different terminal programs, as described below.

2.3.1 The MODCOMP Boot Storage Program BOOTSV

This paragraph describes first the protocol used between the development terminal and the MODCOMP for transferring a bootable MEGATEK program, and then discusses the operation of the BOOTSV program. The BOOTSV program is used both for transferring the TMS terminal program and for transferring various MEGATEK-supplied diagnostic programs which are available only on floppy disks.

2.3.1.1 File Transfer Protocol. Programs from the development terminal are transferred to the MODCOMP using a simple protocol. The NOVA writes on the bus a series of 33-word segments, in which the first word is a sequence number for the segment, and the following 32 words are part of the program being sent. The sequence numbers are positive integers beginning with one. The MODCOMP keeps track of the sequence numbers it receives, and, after each segment, replies with a one-word message consisting of the highest sequence number successfully received. If the MEGATEK fails to receive an acknowledgement or if the acknowledgement is not the same as the number of the last segment transmitted, the MEGATEK retransmits the last segment. The MEGATEK transmits a dummy segment with a sequence number of -1 when it has finished sending the program.

The absolute, or bootstrap, programs sent by the development terminal to the MODCOMP follow a simple format, as illustrated in Figure 2.3.1-1. The first 512 words are always a small loader program. The following 127 words are zero, and the next word is the number of words in the absolute program to be transferred. The next N words (where N is a multiple of 128) contain the absolute program (possibly padded with zeros). Finally, 128 identical words, each of which is the NOVA instruction for a jump to location 2, are sent. Paragraph 2.3.2 explains how this structure is used during bootstrapping of a terminal.

2.3.1.2 Operation of BOOTSV. Figure 2.3.1.2-1 gives a logic flow diagram of BOOTSV. The program utilizes logical device DB9 as the storage area for the boot program, and assumes DB9 to be assigned to an actual disk file large enough to contain the entire boot program. BOOTSV begins by preformatting every sector of the file so that each sector (128 words) contains 127 words of zeros followed by a word containing the one-origin sector number.

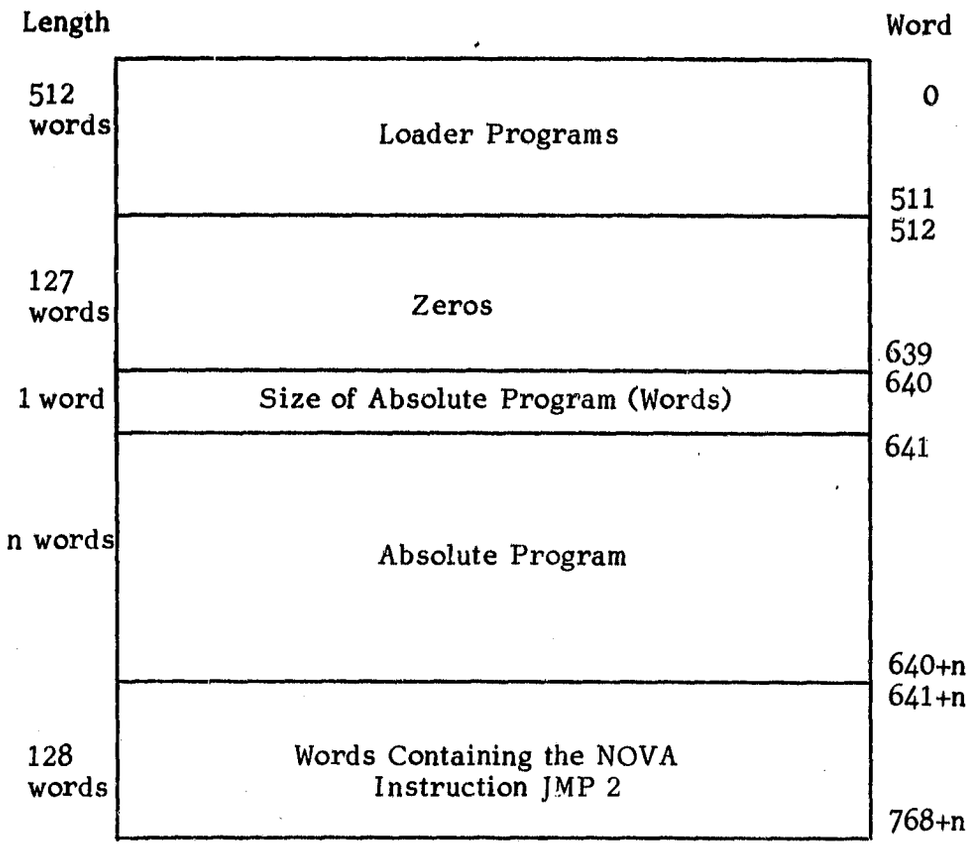


Figure 2.3.1-1
Format of Bootstrap Program

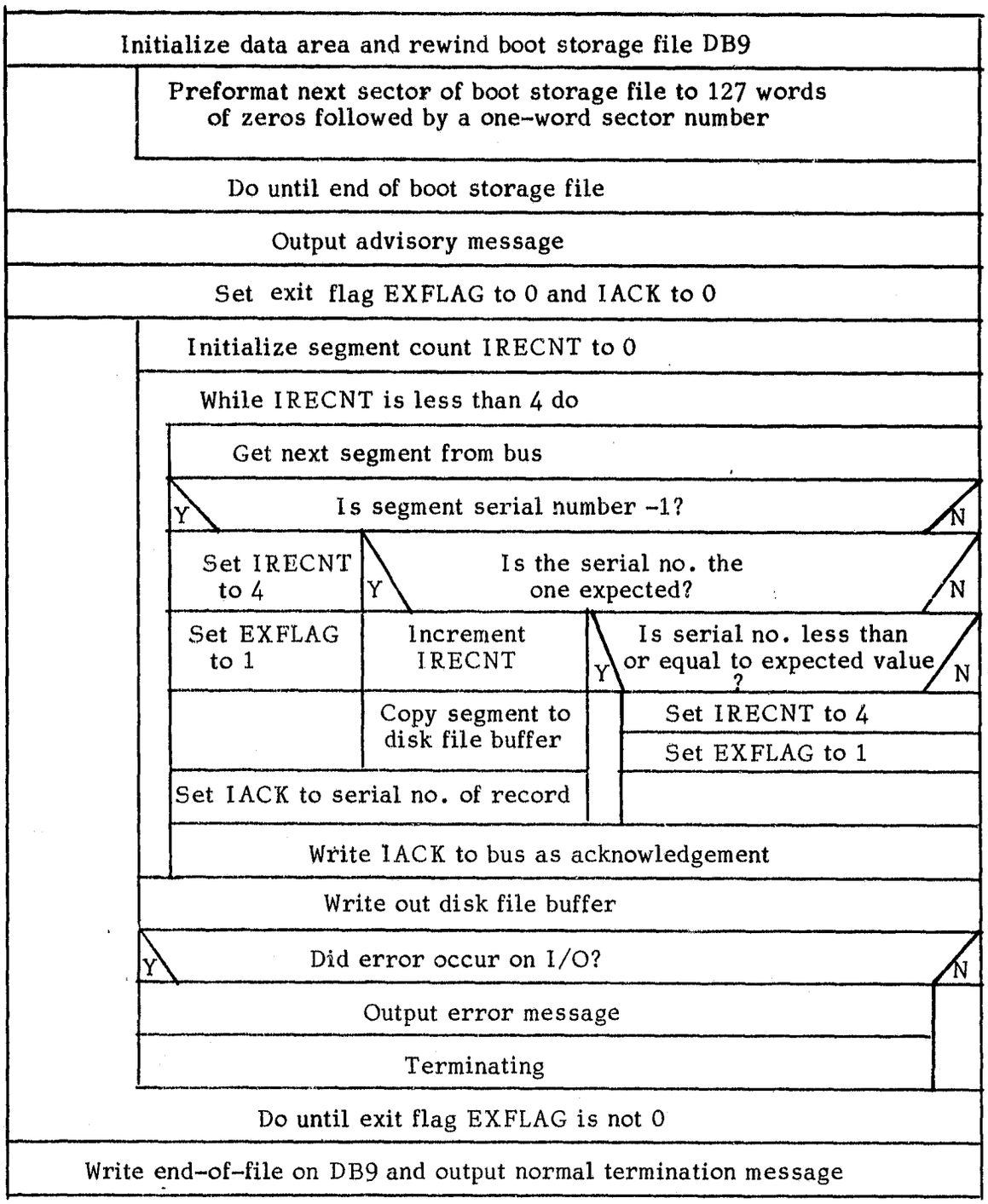


Figure 2.3.1.2-1

MODCOMP Boot Storage Program
(BOOTSV)

BOOTSV then reads segments from the bus according to the protocol described in paragraph 2.3.1.1, blocks the segments into 128-word units, and writes them to logical device DB9. BOOTSV reads from logical device NO0 which is normally assigned to the MEGATEK development terminal. When BOOTSV receives a segment with a sequence number of -1, it terminates normally.

Procedures for using BOOTSV, together with typical file assignments, are given in [12].

2.3.2 The MODCOMP Terminal Booting Program BOOT

The MODCOMP program BOOT reads a MEGATEK bootstrap program (formatted as in Figure 2.3.1-1) from a disk file and writes the bootstrap onto the bus system. BOOT takes its input from logical file DB9 (which can be assigned to any physical MODCOMP file containing a boot program) and reads until either an end-of-file or a sector matching the preformatting (see paragraph 2.3.1.2) is found. BOOT directs its output to logical device NI0, which can be assigned to any MEGATEK terminal on the bus. Detailed procedures for operating the BOOT program are found in [12].

Figure 2.3.2-1 gives a flow diagram for BOOT and shows that the program is simply a utility to copy from a disk file to the bus. At the MEGATEK, when the reset and program load switches are toggled, a 32-word loader program is read into the MEGATEK from a Read Only Memory (ROM) and the terminal's NOVA is started. This small program reads in the 512-word loader at the beginning of the bootstrap (see Figure 2.3.1-1), and this second loader reads the next 128 words to determine the length of the program element. The second loader initiates a DMA transfer for the length of the program plus 128 words, places a JMP * (jump to the current location) instruction in the next 128-word interval following where the program will lie, and transfers control to the JMP * instruction. The last word read in by the DMA command will overlay the JMP * instruction with a JMP 2 command, and the computer will then jump to location 2 to begin execution of the loaded program.

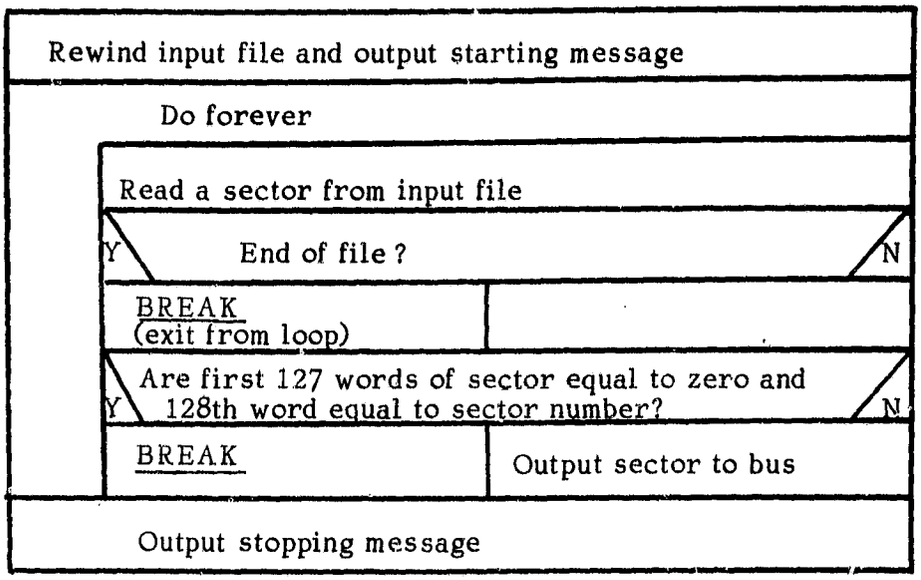


Figure 2.3.2-1

MODCOMP Terminal Boot Program
(BOOT)

SECTION III
NOVA SOFTWARE

3.0 INTRODUCTION

The principal communications software in the NOVA computers (which are the major active component in the TMS MEGATEK terminals) are the bus handler programs TBUS and GBUS. One additional program, UPMAIN, is used in the transfer of bootable absolute programs from the development MEGATEK to the MODCOMP; UPMAIN is the companion program to BOOTSV (described in paragraph 2.3.1). The following paragraphs describe this NOVA software.

3.1 The NOVA Bus Handler Programs TBUS and GBUS

The program TBUS is used with the TMS graphics terminal program (see reference [5]), while the routine GBUS is used with the boot transfer program UPMAIN (paragraph 3.2). TBUS and GBUS each attempt to "sign on" (establish a logical bus connection) to the TMS MODCOMP when they are first activated, but the two routines differ in their action if the TMS system is not available. TBUS calls a graphics routine in the terminal to output a message and waits for the user to specify an alternate logical link; GBUS, on the other hand, simply continues to retry the establishing of the link to TMS until it is successful. This difference in behavior was chosen to avoid requiring that a number of graphics routines be included in any NOVA utility program which needs access to the bus. Ordinarily, such utility programs are not run unless TMS is known to be operating.

Other than this difference, TBUS and GBUS are identical and the remaining discussion of the bus handler refers to both routines.

The following paragraphs present the NOVA bus handler in a discussion which has a structure parallel to the description of the MODCOMP Bus Symbiont in Section II. The NOVA handler, however, is much simpler than the MODCOMP symbiont because the NOVA routine deals with a simpler

operating system, and because the NOVA bus interface program has a simpler user interface. These differences are considered more fully in the following paragraphs.

The NOVA bus handler is composed of three user-callable entry points (which are used to queue read, write, and reset operations), plus a handler for interrupts from the BIU. Paragraph 3.1.1 discusses briefly the interrupt handler's interface to the operating system, while paragraph 3.1.2 describes the user task interface to the entry points. Paragraph 3.1.3 considers multitasking operation with the bus, and paragraph 3.1.4 provides detailed information about the protocol between the NOVA and its BIU.

3.1.1 NOVA Operating System Interfaces

The NOVA bus handler is designed to operate with either the Diskette Operating System (DOS - [15]) or the Real Time Operating System (RTOS - [16]) provided by Data General Corporation. Both of these operating systems permit a user-written handler (as opposed to vendor-supplied handler) to be inserted easily into a program by the .SYSTM call .IDEF. After having been inserted by this .SYSTM call, a user handler is automatically given control whenever the device for which it is registered interrupts the NOVA. The handler is given full physical control over the device and can interrogate its status directly and issue I/O commands directly (the status and commands do not pass through the operating system).

TBUS and GBUS take advantage of these facilities by including a section of code which is registered as a user interrupt handler.

3.1.2 User Task Interfaces

A NOVA user task communicates with the bus through FORTRAN or assembly language subroutine calls to the routines RSET, RBUS, and WBUS. All of these routines are "wait mode" (the calling program does not receive control again until the requested operation has been completed).

RESET is a parameterless subroutine which is used by an application task to reset the NOVA BIU by pulsing the BIU's hardware reset line; this has the effect of reinitializing the BIU and causing it to discard any messages currently held in its buffers. Furthermore, the logical link which exists (if any) between the NOVA BIU and any other network BIU is broken by the reset activity; a logical link must then be established before further traffic can be successfully sent on the network. For more details on establishment of logical links ("signing on"), consult reference [4].

The call to read information from the bus is of the form

CALL RBUS (Buffername, Bufferlength)

where Buffername is the address of the first word of the buffer into which information is to be read, and Bufferlength is an integer variable giving the maximum number of words to be read. A user task is then suspended until completion of the read. The requested physical read is not actually issued to the BIU until the BIU indicates that there are data waiting for the NOVA. The read is then performed and is considered completed when the BIU has transferred all the words it has queued for the NOVA, or when the requested word count has been reached, whichever occurs first. When the user task is reactivated after completion of the read, the actual number of words read (up to the limit specified in the call) is returned in the location Bufferlength.

To write information onto the bus, a user task first sets the one word of the common block BUSCOM to contain the number of words to be transferred. The task then issues a call of the form

CALL WBUS (Buffername)

where Buffername is the address of the first word of the output buffer.

3.1.3 Multitasking Operation

The NOVA bus handler is structured to operate within a multitasking environment, in which a single user program may consist of a number of separate activities running logically asynchronously, but all referencing the same copy of the bus handler code. The NOVA operating systems do not

support multiprogramming, and if more than one copy of the bus handler code were simultaneously in use, the bus interface would not work properly.

In the multitasking environment, the bus handler insures that at most one physical operation is outstanding to the bus interface at any time, and further allows at most one user task read and one user task write to be candidates for execution at any time. The NOVA operating system's semaphore capability is used to block other tasks which request bus I/O until previous operations of the same type are complete.

Whenever any user program which accesses the bus is started, at the first bus operation the BIU is reset by the bus handler and certain other initialization is performed. This initialization must be performed regardless of whether the first operation is a call to RSET, to RBUS, or to WBUS, and the initialization must furthermore be done only once. The bus handler is consequently structured so that at the beginning of each of the user-callable subroutines is a call to an internal initialization subroutine SIGNON. When this subroutine is first used, it modifies itself so that it returns immediately from all future calls. This method was chosen to provide a low-overhead safe method of mutual exclusion.

3.1.4 NOVA/BIU Protocol

The NOVA in a MEGATEK terminal communicates with its BIU through a DMA interface which is described in [3]. The following discussion gives design and implementation details for the supporting protocol.

Paragraph 3.1.4.1 contains information about the DMA interface, while paragraphs 3.1.4.2 and 3.1.4.3 discuss design requirements and traffic formats for the interface, respectively. Protocol signals are described in paragraph 3.1.4.4, and the protocol state diagram is discussed in paragraph 3.1.4.5. Paragraph 3.1.4.6 deals with the significant events in the protocol. A detailed description of the interface protocol appears in paragraph 3.1.4.7, and flow diagrams of the NOVA bus handler are given in paragraph 3.1.4.8.

3.1.4.1 The NOVA DMA Interface. The NOVA is connected to the BIU through a Data General Model 4040 general purpose interface board (see [17]), to which interface circuitry (see [3]) has been added. Data are transferred over this interface in 16-bit-parallel DMA mode. A full complement of handshaking signals is present, so that both the NOVA and the BIU can control the speed at which the other party is sending data across the interface. Because the transfers are DMA-operated, this handshaking is transparent to the NOVA communications software and is therefore not discussed further here.

Physical I/O instructions issued to the DMA interface are interpreted by circuitry on the 4040 board. The board has three registers -- the A, the B, and the C registers -- which are used to control I/O operations. These registers have the general functions shown in Table 3.1.4.1-I (more details are given in [3]), and are addressed by the NOVA I/O instructions DIx and DOx, for inputting from and outputting to the registers, respectively, where the x is one of the register designators (A, B, or C).

Table 3.1.4.1-I
NOVA DMA Register Usage

Register	Meaning when Read (Referenced with DIx)	Meaning when Written (Referenced with DOx)
A	Status of interface	I/O command
B	Next memory address to be written	First memory address to be read or written
C	Two's complement of number of words remaining to be transferred	Two's complement of number of words to be transferred

Also visible to the BIU as a prt of the interface is the state of the "booted" flip-flop on the 4040 board. This flip-flop can be set by the NOVA through an ordinary I/O operation to the board (see [3]). The flip-flop is used, as described in paragraph 3.1.4.4 in bootstrapping the MEGATEK terminals. In addition to these control registers, the standard NOVA BUSY and DONE interface signals play a part in the bus interface. The BUSY signal is asserted by the 4040 board whenever an I/O operation to the board is in progress; the signal is removed when the word count specified in the DMA transfer is satisfied. It should be noted that during a NOVA write the BUSY signal is removed as soon as the last word is transferred to the 4040 board, which occurs before the last word is taken by the BIU.

The DONE flip-flop serves as an interrupt request flag for the NOVA; this signal is set by the 4040 board whenever the word count in a transfer is satisfied. The DONE signal can also be set by the BIU to cause an interrupt of the NOVA.

3.1.4.2 Interface Design Requirements. The requirements for the NOVA/BIU protocol are essentially the same as those for the MODCOMP/BIU interface (paragraph 2.1.4.2). The protocol must permit full handshaking so that either the NOVA or the BIU can limit the flow of data to accommodate its own limited buffer space.

Speed of data transfer is important between the NOVA and its BIU. Because of the structure of the TMS, however, where a choice arises, input into the NOVA should be favored over output from the NOVA. This choice is important in the TMS for two reasons. First, a single MODCOMP computer (coupled to a single BIU) serves a number of user terminals, and second, the volume of traffic from the host to the terminals is from 100 to 1000 times as great as the volume of traffic from the terminals to the host. Consequently, in order to avoid choking the host with data, the terminals need to be able to accept and dispose of data from the host as fast as possible.

The interface protocol must also be able to support bootstrap loading of the NOVA in a MEGATEK terminal, as described in paragraph 2.3.2. Bootstrapping of the NOVA is initiated by the NOVA's program load ROM, which performs bus I/O differently from the way the TMS terminal program operates. First, the ROM loader uses slightly different I/O commands, and second, the loader is incapable of handling interrupts from the BIU.

Because of these differences, then, the BIU must handle boot program transfers differently from the way in which it handles data transfers, and the BIU must consequently be able to detect when the NOVA is requesting a boot program. This detection is handled through the use of the "booted" flip-flop described in paragraph 3.1.4.4. Furthermore, the BIU must be able to distinguish between incoming network packets which are data and incoming packets which are parts of a boot program.

This latter requirement is met by causing the NOVA BIU to respond to two network addresses (a "boot" address and a "data" address, each unique to the BIU). Only limited discussion of his two-address operation is contained in the following paragraphs; more information is given in [4] and [6].

3.1.4.3 Interface Traffic Format. Unlike the MODCOMP/BIU interface protocol, the protocol between the NOVA and its BIU does not provide for the passing of network packet headers between the BIU and the NOVA. The BIU regards data from the NOVA as a continuous stream of words. The BIU buffers the words until either the NOVA stops offering data to the BIU or until 60 words of data have been received without a pause from the NOVA. At that point, a packet is queued for output to the network. Data from the net for the NOVA are also buffered, and are treated as a continuous stream of words which are given to the NOVA one-by-one as the NOVA requests them.

The packet format of Figure 2.1.4.3-1 is, of course, used for all packets emitted to and received from the network by the NOVA BIU. The Destination Address (DA) field of the packet header is filled in by the NOVA BIU either to be a broadcast address (for BIU status messages - see [4]) or to be the network address of the other BIU to which the NOVA BIU is currently "signed on" (for all other messages).

3.1.4.4 Protocol Signals. There are three bits in the BIU status word which are used by the NOVA bus interrupt handler to determine the state of the interface. These bits are OUTPUT READY (bit 15 [low-order bit] of the status word), INPUT READY (bit 14), and BIU ERROR (bit 13). Other bits of the status word are not used.

The INPUT READY bit is set when the BIU is holding data which is to be transferred to the NOVA, and the OUTPUT READY bit is set whenever the BIU is ready to accept data from the NOVA. The BIU ERROR bit is not presently used in the interface protocol.

The BIU determines the status of the interface by examining the word-by-word handshaking lines, by looking at the BUSY and DONE signals, and by observing the "booted" flip-flop on the 4040 board. When the BIU detects that the NOVA is offering a word of data, the BIU assumes that a NOVA output operation is in progress and accepts words from the NOVA until the NOVA has completed its write. When the NOVA asserts both the BUSY signal and the "word-desired" signal (and not the "word-offered" signal), the BIU recognizes that a NOVA input operation is in progress and passes words to the NOVA until either the BIU has transferred all the data it has or the NOVA input operation ends. When the BIU exhausts its supply of input words for the NOVA, it examines the DONE flag before interrupting the NOVA; the BIU interrupts only when the DONE flag is not asserted so that its interrupt is not lost because of a previously requested interrupt (all interrupt requests are signaled by asserting DONE). Descriptions of how these signals are used in normal operation are found in paragraph 3.1.4.7.

The BIU examines the "booted" flip-flop to determine whether or not the NOVA is executing a program which is capable of accessing the bus.

3.1.4.5 Protocol State Diagram. Figure 3.1.4.5-1 shows the states of the NOVA/BIU protocol together with allowable transitions between the states. The conditions leading to each transition and the consequent actions which occur are described in the decision table shown in Table 3.1.4.5-I. The protocol events are described more fully in paragraph 3.1.4.6 and a detailed description of protocol operation is given in paragraph 3.1.4.7.

Four states of the protocol are illustrated. The "Booting" state means that the BIU is accepting packets addressed to its secondary (boot) network address, and that the BIU is expecting that the information it is receiving is a program for the terminal to execute.

The "Idle" state means that no I/O operation is in progress to transfer data between the NOVA and its BIU.

In the "NOVA Input" state, data are being transferred from the BIU to the NOVA, and in the "NOVA Output" state, data are being transferred in the opposite direction. Further information about these states and about the transitions among them is given in the following paragraphs.

3.1.4.6 Protocol Events. The normal idle state of the NOVA/BIU interface is for neither participant to have an operation in progress. When the interface is active, an I/O operation is underway to transfer data in exactly one direction -- either from the NOVA to the BIU(NOVA output or from the BIU to the NOVA (NOVA input). Occurrences which cause the interface to leave the idle state or which affect an active state are termed events.

An "event" in the NOVA/BIU protocol occurs when one of the following four things happens:

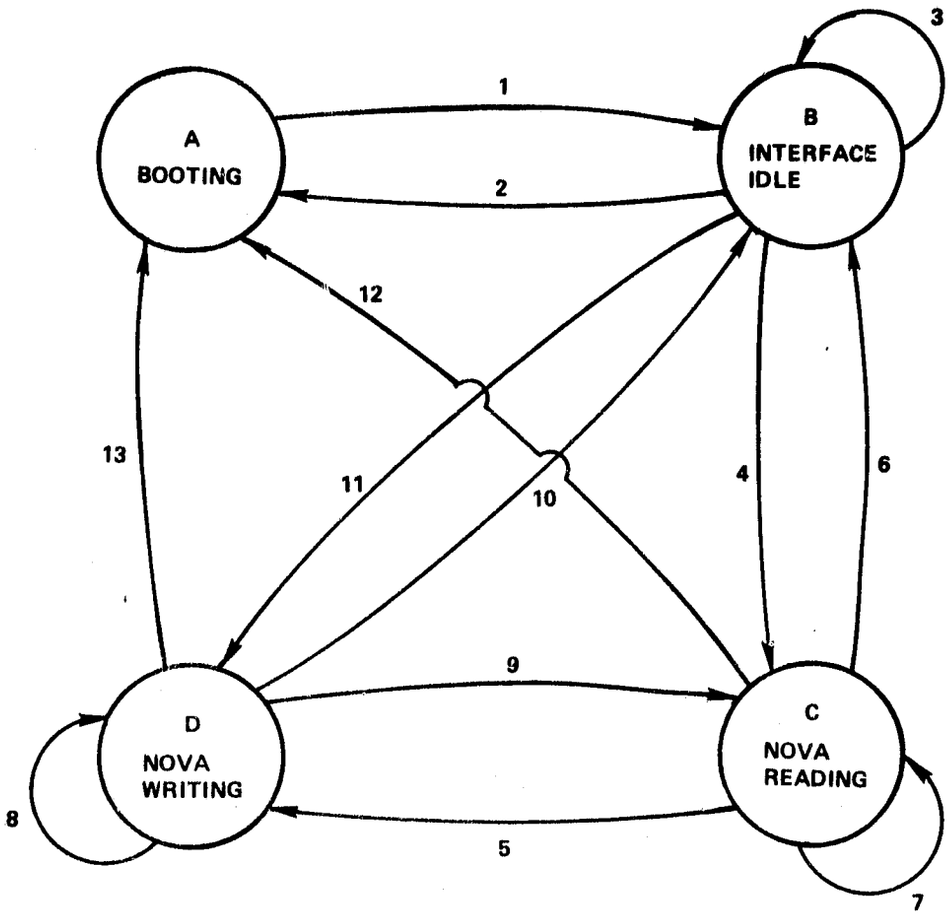


Figure 3.1.4.5-1

NOVA/BIU Protocol State Diagram

Table 3.1.4.5-I

Explanation of NOVA/BIU Protocol State Diagram Transitions

Transition	1	2	3	4	5	6	7	8	9	10	11	12	13
Conditions													
a. Handler activated [1]	Y	–	Y	Y	Y	Y	Y	Y	Y	Y	Y	–	–
b. Possible to Start Read [2]	–	–	N	Y	–	–	Y	–	Y	N	N	–	–
c. Possible to Start Write [3]	–	–	N	–	Y	–	–	Y	–	N	Y	–	–
d. "Booted" Flag Set	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N
Actions													
a. Handler starts read				X			X		X				
b. Handler starts write					X			X			X		
c. Handler awaits next event (BIU interrupt or user task call for read or write			X			X				X			
d. BIU stops listening to network "boot address and starts listening to "data" address	X												
e. BIU stops listening to network "data" address and starts listening to "boot" address		X										X	X

NOTES:

1. Transitions can occur only when the booted flag is reset (either by front panel switches or by program control) or when certain parts of the NOVA bus handler are activated. Activation of the handler can occur by the following ways:

   a. Interrupt of the NOVA by the BIU
   b. Interrupt of the NOVA by the DMA board (transfer count satisfied)
   c. Call of RBUS or WBUS by user task (call to RSET resets BIU but does not in itself cause a transition)

2. "Possible to Start Read" means that the handler has a read operation queued (or is activated by an RBUS call) and the BIU INPUT READY signal is asserted.

3. "Possible to Start Write" means that the handler has a write operation queued (or is activated by a WBUS call) and the BIU OUTPUT READY signal is asserted.

1. The NOVA bus handler issues an operation to the interface.

2. The BIU receives bus traffic for the NOVA when the interface is idle.

3. An outstanding operation is ended in one of these ways:

   (a) the requested number of words are transferred

   (b) the operation is interrupted by the BIU

4. The state of the "booted" flip-flop on the 4040 board changes.

The first class of events occurs only when the interface is idle. Whenever a user task calls the RBUS, WBUS, or RSET entry points of the bus handler, the handler checks the status of the interface, and if the interface is idle, the handler may initiate an operation. Similarly, when an operation ends, the bus interrupt handler, having serviced the interrupt, knows that the interface is now idle and checks to see whether any queued operation can be started. A NOVA input operation is not started unless the BIU indicates in its status word that the BIU has data for the NOVA, and a NOVA output operation is not started unless the BIU indicates that it is ready to accept output.

The second class of events occurs when a network packet is received by the BIU, but no operation is in progress on the interface. The BIU then sets the status word bit to indicate that data are available for the NOVA, and then interrupts the NOVA so that the bus interrupt handler will be awakened. The interrupt handler can then issue a read operation if one is queued (the issuance of the operation is an event of the first class above).

An event of the third class occurs normally on NOVA output when the stated word count to transfer is exhausted and the 4040 board consequently interrupts the NOVA. On NOVA input, the operation may be ended either by satisfaction of the word count or by the BIU's transferring of the last word available to the NOVA. In either case, an interrupt to the NOVA occurs. If an operation is ended by an interrupt in any other case, an error condition has occurred.

The fourth category of events deals with the program being executed by the NOVA. If the NOVA is to use the bus but does not contain an executable program in its memory, the "booted" flag is reset, and the BIU knows to accept input from its "boot" network address and to treat the information as a program to be loaded. (The "booted" flag is reset when the NOVA is powered on or reset from the front panel switches, and can also be reset under program control through the handler's RSET entry point.)

When a program load into the NOVA is completed, the program begins execution and promptly sets the "booted" flag so that the BIU knows that the program is executing, and that no further executable code should be accepted from the network "boot" address until the "booted" flag is again reset. Further traffic with the bus is handled through the BIU's "data" network address.

Each of the first three classes of events causes the NOVA bus handler to be invoked, while the fourth may or may not invoke the handler, depending on how the flip-flop state is changed.

3.1.4.7 Protocol Operation. This paragraph describes how the protocol operates under various circumstances. Reference to the protocol state diagram, Figure 3.1.4.5-1, may be helpful.

When the NOVA is initially powered on, the "booted" flag on the 4040 board is forced to its reset state, and the BIU consequently begins to listen to the network at its "boot" address rather than its "data" address.

At this point, the BIU does not expect (and will not accept) output from the NOVA. While the BIU is waiting for boot packets to be received from the net, it periodically polls the "booted" flag in case an executable program is loaded to the NOVA by some means other than the bus (such as from a floppy diskette on the development terminal). As packets of boot information come in, they are passed to the NOVA over the interface using the normal word-by-word handshaking. When the BIU notices that the "booted" flag is now asserted by the NOVA, the protocol makes a transition to the idle state.

In the normal idle state, neither the BUSY nor the DONE signals are asserted, the "booted" signal is asserted, the OUTPUT READY bit in the BIU status word is set, and the NOVA "waiting for input" handshaking line is asserted (this last condition is a quirk of the hardware, as described in [3]). The interface will stay in the idle state indefinitely until either input arrives to the BIU from the network, an I/O operation is started by the NOVA bus handler, or the state of the "booted" flag is changed.

When input arrives from the network for the NOVA, the BIU sets the INPUT READY bit in the status word (paragraph 3.1.4.4). If neither the BUSY nor the DONE signal is asserted, the BIU then interrupts the NOVA.

Within the NOVA, when a user task calls the RBUS entry point to read from the bus, if the interface status is "idle" and the BIU's INPUT READY bit is set, a DMA read operation is started for the number of words specified in the RBUS call. If either of these conditions is not true, the request is queued by the handler and the calling task is blocked.

When a user task calls the WBUS entry point, if the interface status is "idle" and the BIU's OUTPUT READY bit is set, a NOVA output operation is started to transfer the number of words stated in the call. Otherwise, the write request is queued and the calling task is blocked.

When a user task calls the RSET entry point, an operation is immediately issued to reset the BIU.

Interrupts of the NOVA occur under the following three conditions:

1. When the word count specified in an I/O operation is satisfied.
2. When, on input of data, the BIU transfers all the words which it currently has buffered for the NOVA.
3. When input from the net is received by the BIU and the interface state is "idle".

In any of these cases, the NOVA bus handler checks the previous interface state and unblocks the user task which was waiting on the now-completed I/O if the interrupt occurred under conditions 1 or 2. The handler then checks to see whether an input operation can be started (a user task read must be queued and the BIU INPUT READY bit must be set), and if so, the operation is initiated. If an input operation cannot be begun, the handler checks to see whether an output operation can be started, and, if so, begins a DMA write. These checks to see whether operations can be started are also performed when interrupts occur under condition 3.

3.1.4.8 Flow Diagram of the NOVA Bus Handler. The preceding paragraphs have discussed the broad operation of the NOVA/BIU protocol, and have indicated the general structure of the NOVA bus handler. Figures 3.1.4.8-1 through 3.1.4.8-5 are flow diagrams of the user-callable entry points RBUS, WBUS, and RSET, the internal initialization routine SIGNON, and the bus interrupt handler, respectively. The routine SIGNON is shown as it appears in the TBUS handler.

The operation of the various parts of the NOVA bus handler is coordinated principally by the NOVA operating system's semaphore capability. As can be seen from the flow diagrams, the semaphores READER and WRITER are used to insure that only one user task at a time presents a read or write operation, respectively, for the bus handler to consider. Other tasks wishing to perform the same operation as presently being serviced are blocked, or

RBUS (buffername) [common block BUSCOM contains length]

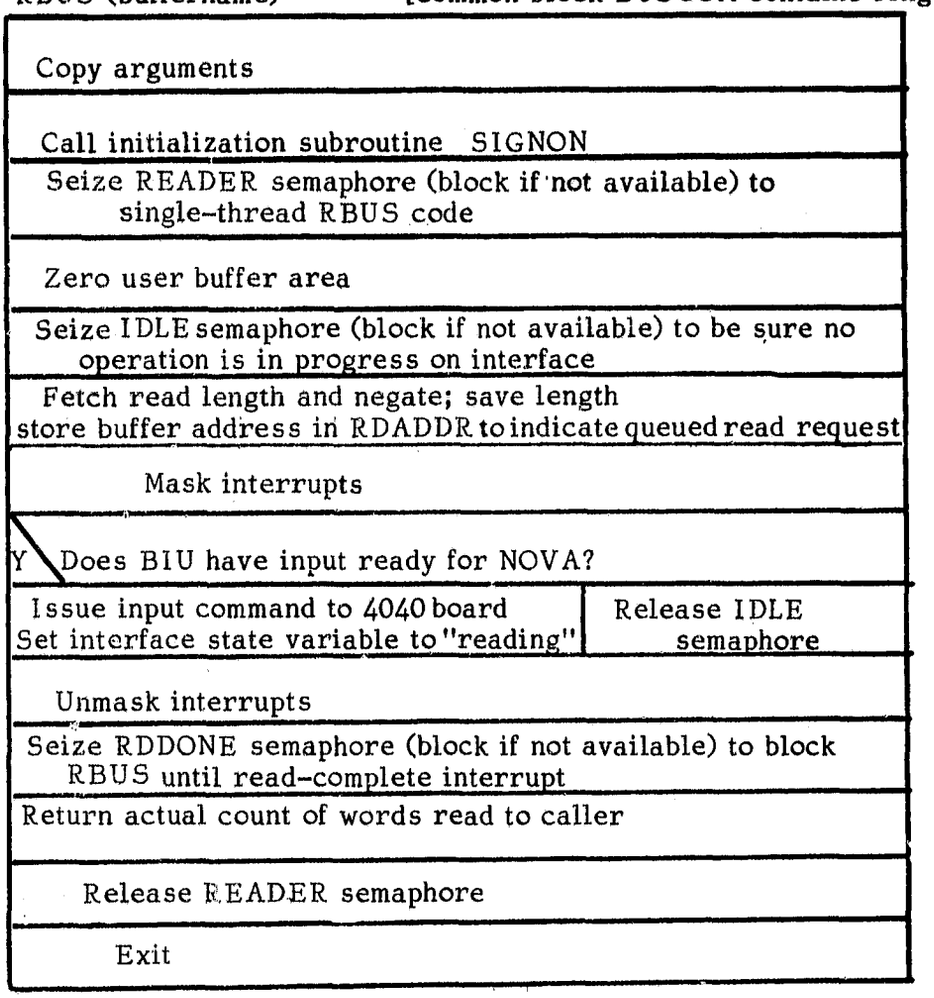


Figure 3.1.4.8-1

NOVA Bus Handler Read Routine RBUS

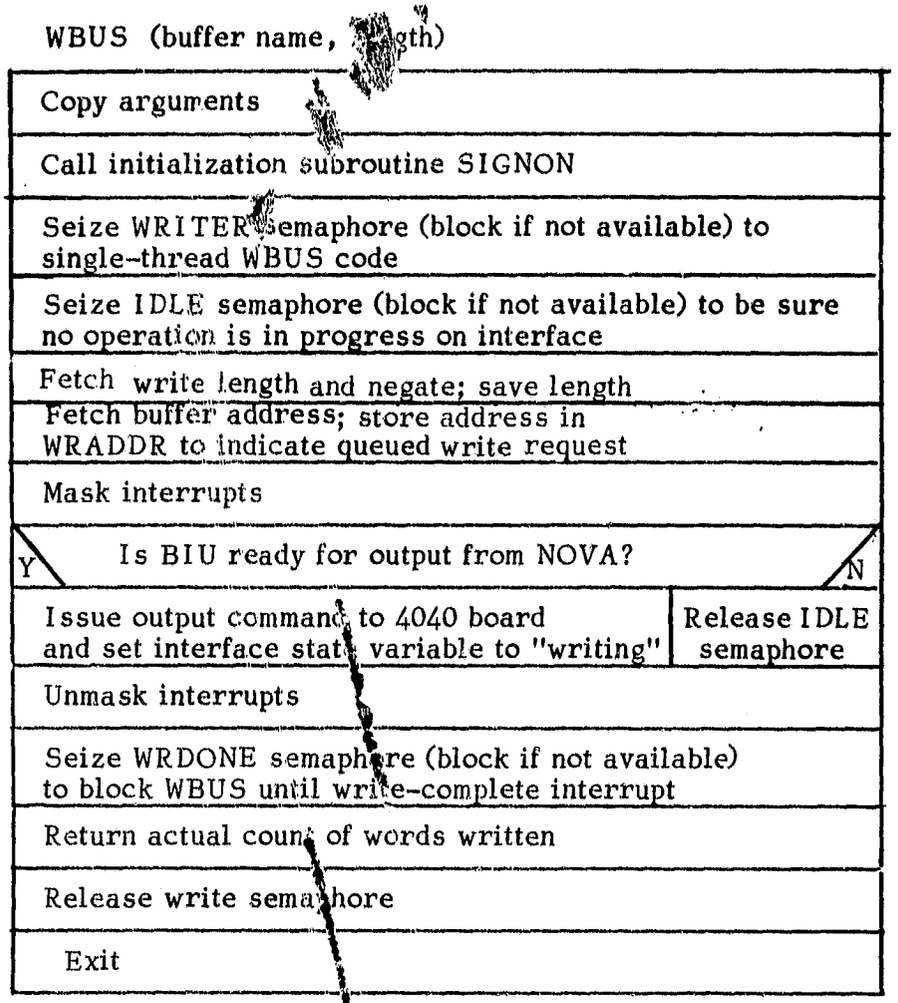


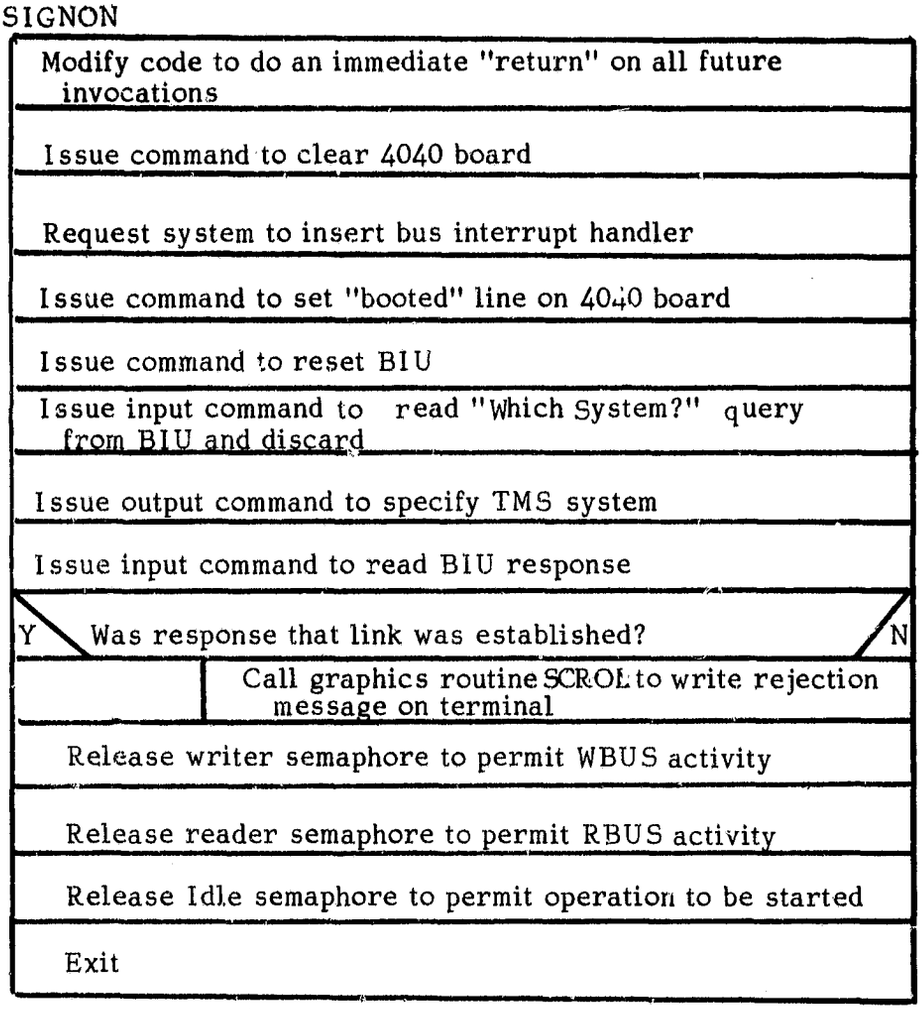
Figure 3.1.4.8-2
NOVA Bus Handler Write Routine WBUS

RSET

Copy arguments
Call initialization subroutine SIGNON
Seize IDLE semaphore (block if not available) to be sure no operation is in progress on interface
Issue command to reset BIU
Wait for completion of reset
Release IDLE semaphore
Exit

Figure 3.1.4.8-3
NOVA Bus Handler BIU Reset Routine RSET

SIGNON



NOTE: This code is self-modifying since the technique offers the easiest way of insuring that the code will be executed only once in a NOVA multitasking environment.

Figure 3.1.4.8-4

NOVA Bus Handler Internal Initialization Routine SIGNON

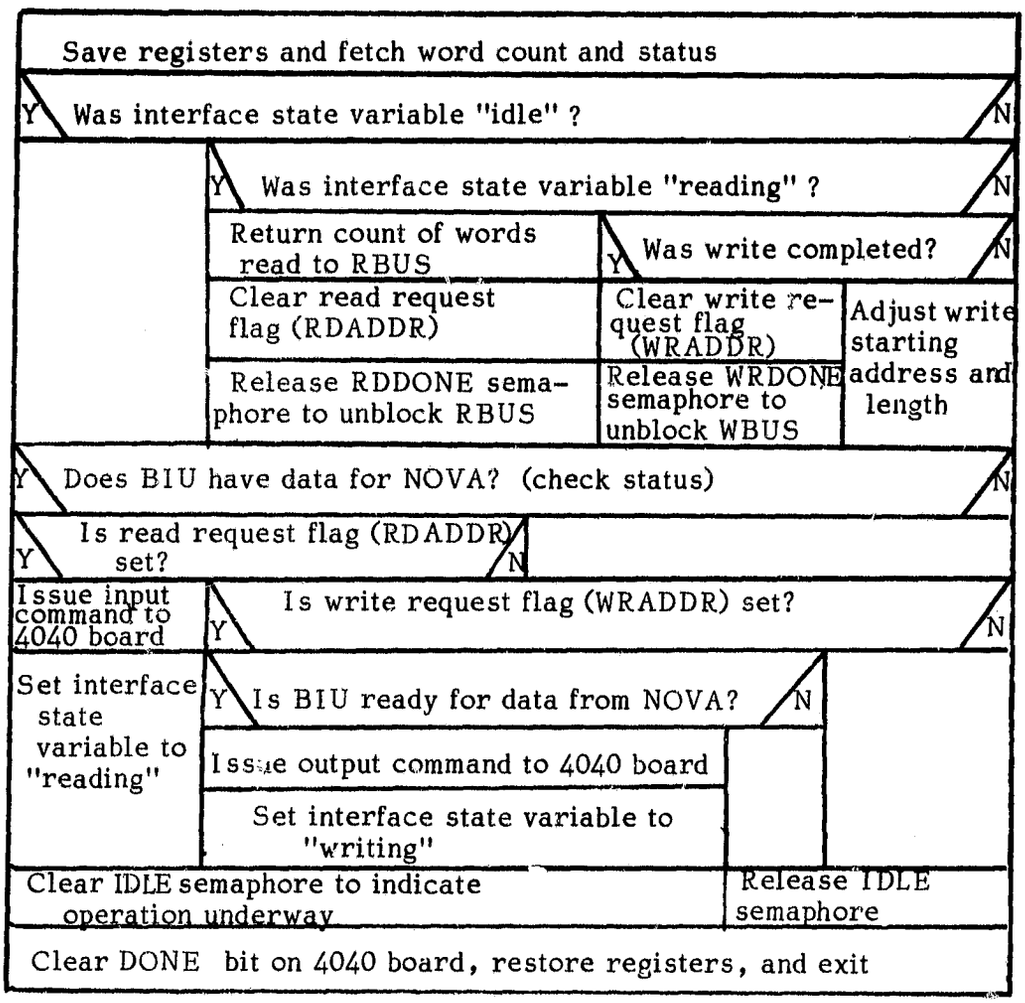


Figure 3.1.4.8-5
NOVA Bus Interrupt Handler

suspended, until the previous requests are completed. The semaphore IDLE is used to insure that at most one I/O operation at a time (either a read or a write) is issued to the DMA interface; IDLE serves to coordinate the independent activity of the RBUS and WBUS routines.

3.2 The NOVA File Transfer Program UPMAIN

Paragraph 2.3 discussed the TMS requirement to transfer files containing absolute (boot) programs from the development MEGATEK terminal to the MODCOMP, and from the MODCOMP to any TMS terminal. The MODCOMP program to receive these file transfers, BOOTSV, was discussed in paragraph 2.3.1. This paragraph discusses the NOVA program UPMAIN, which sends the boot file to BOOTSV over the bus system.

The NOVA file transfer program UPMAIN is actually a multitasking program composed of the main routine, UPMAIN, which performs initialization and activates the task RDR before beginning to listen for input from the bus, and the daughter task RDR. RDR constructs the boot program in the format shown in Figure 2.3.1-1 and calls the subroutines SEND and TOMODC to write portions of the program to the bus. The module CLK contains a user clock routine (task timer) which is started each time a segment is written to the bus; the timer provides assurance that the program will not become irretrievably stuck if an acknowledgement is not sent or not received properly. The routine REC is used to simulate semaphores for communication among CLK, TOMODC, and UPMAIN. (Simulation of semaphores was necessary because of problems encountered with using the operating system semaphores concurrently with a user clock.)

Figures 3.2-1 through 3.2-6 are flow diagrams of these program elements.

The multitasking structure of UPMAIN arises from the file transfer protocol described in paragraph 2.3.1.1, in which the boot program is sent from the MEGATEK to the MODCOMP in 33-word sequence-numbered packets which are individually acknowledged by the MODCOMP. At the NOVA, the file transfer program should be able to transmit a segment, therefore,

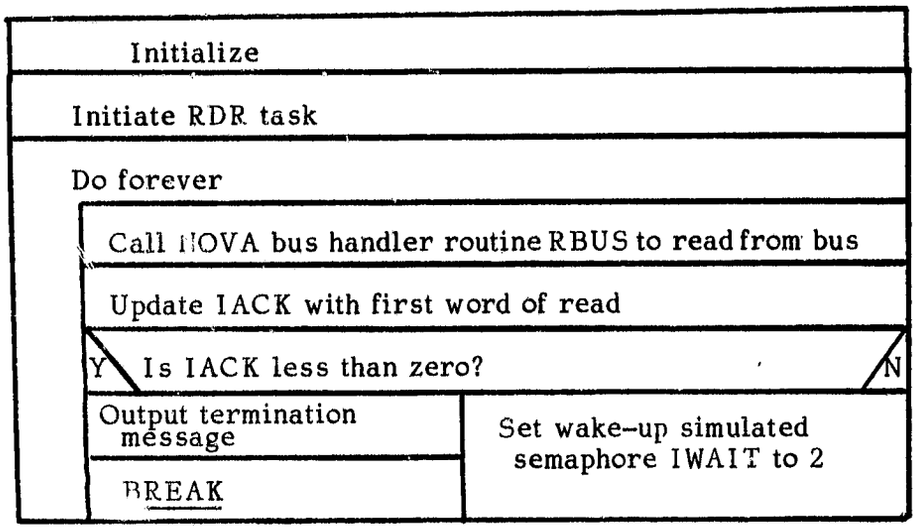


Figure 3.2-1
Main NOVA File Transfer Program UPMAIN

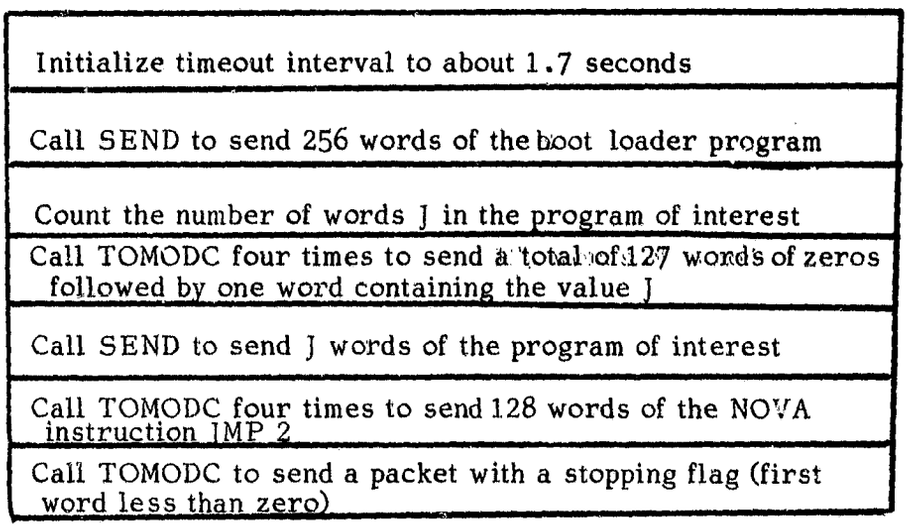


Figure 3.2-2
NOVA File Transfer Task EDR

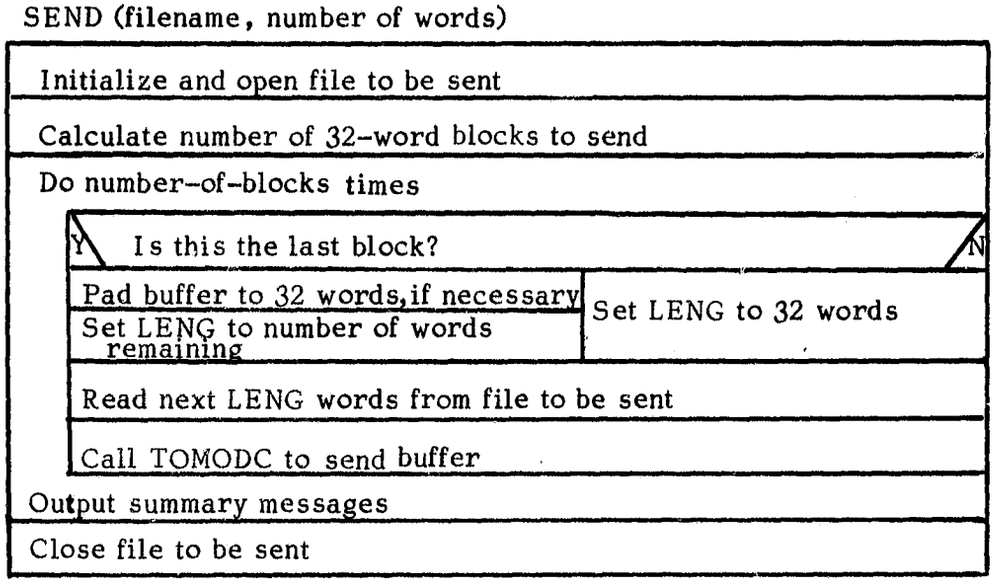


Figure 3.2-3

NOVA File Transfer Subroutine SEND

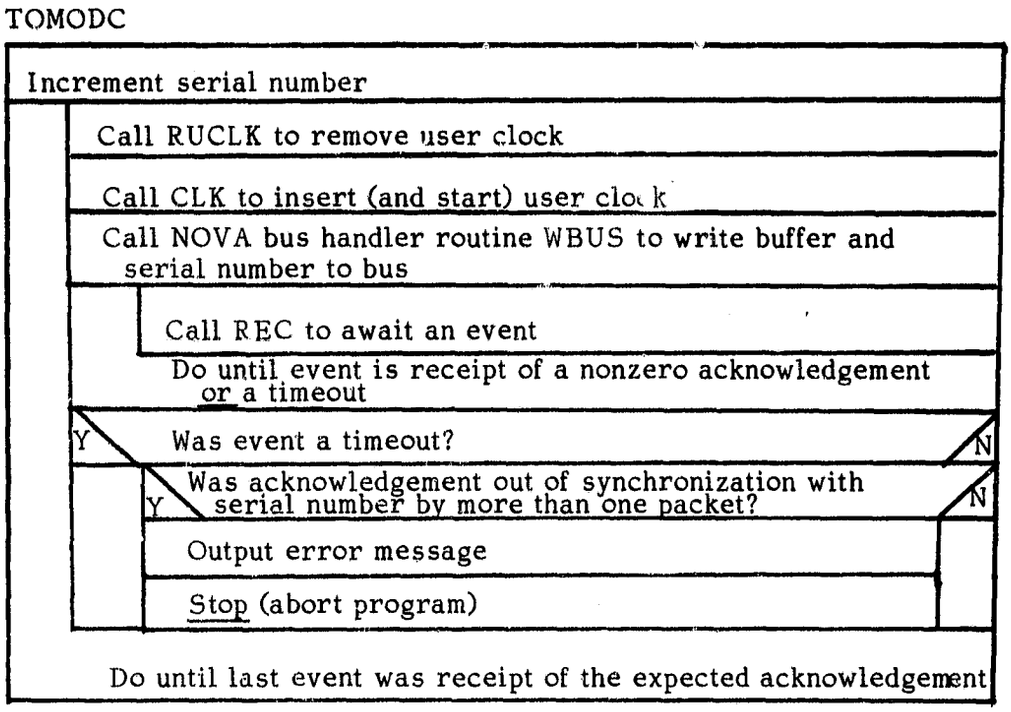


Figure 3.2-4

NOVA File Transfer Subroutine TOMODC

CLK (ICYCLE) – Entry Point

Fetch ICYCLE parameter describing frequency of user clock interrupts
Call operating system function .DUCLK to define user clock

RUCLK – Entry Point

Call operating system function .RUCLK to remove user clock

Clock Interrupt Routine

Set simulated semaphore IWAIT to 1

Figure 3.2-5

NOVA File Transfer User Clock Routine CLK

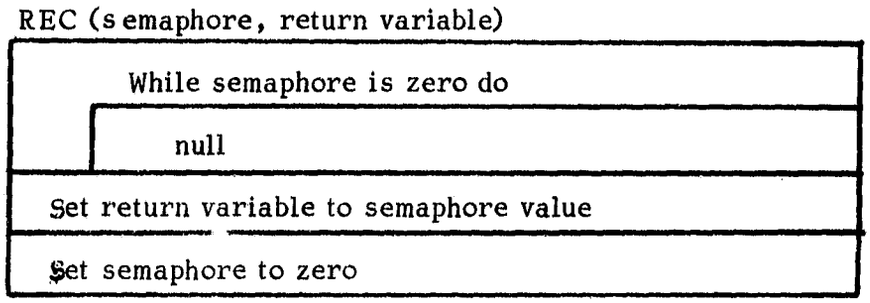


Figure 3.2-6
NOVA File Transfer Semaphore Simulator REC

and block itself until either an acknowledgement has been received, or until a timeout period has elapsed. A convenient method of accomplishing this end is then to allow the RDR task (through its subroutines SEND and TOMODC) to emit a segment to the MODCOMP and then to block itself on the semaphore IWAIT. RDR is reawakened either by the main routine UPMAIN (when input arrives from the bus) or by the user clock routine CLK when a task timer expires. At that point, the subroutine being executed in RDR (SEND or TOMODC) checks to see how it was awakened and if it was awakened by an acknowledgement, whether the acknowledgement is correct. The subroutine can then either retransmit the last segment, transmit the next segment, or terminate, according to the outcome of the checks.

When RDR has completed sending the entire boot program (together with the auxiliary records shown in paragraph 2.3.1), it sends a dummy segment with a sequence number of -1 to the MODCOMP. This sequence number signals the end of the process so that the programs in each computer can terminate normally.

REFERENCES

[1] Hopkins, G. T., A Bus Communications System, The MITRE Corporation, MTR-3515, November 1977.

[2] Roman, G. S., The Johnson Space Center Broadband Communications System, The MITRE Corporation, MTR-3621 (JSC #14495), 1 July 1978.

[3] Brown, J.S. and Weinrich, S. S., Trend Monitoring System (TMS) Communications Hardware - Volume I - Computer Interfaces, The MITRE Corporation, MTR-4721 (JSC #14682), February 1979.

[4] Brown, J. S. and Hopkins, G. T., Trend Monitoring System (TMS) Communications Hardware - Volume II - Bus Interface Units, The MITRE Corporation, MTR-4721 (JSC #14723), March 1979.

[5] Brown, J. S., Trend Monitoring System (TMS) Graphics Software, The MITRE Corporation, MTR-4725 (JSC #14795), April 1979.

[6] Gregor, Paul J., Trend Monitoring System (TMS) Communications Software - Volume II - Bus Interface Unit (BIU) Software, The MITRE Corporation, MTR-4723 (JSC #14793), April 1979.

[7] Brown, J. S. and Lenker, M. D., Diagnostic Procedures for Trend Monitoring System (TMS) Communications, The MITRE Corporation, MTR-4724 (JSC #14794), April 1979.

[8] MAX IV Technical Manual - Volume I - General Operating System, Modular Computer Systems, Inc., Publication 220-61034-000, May 1977.

[9] Technical Manual - MODCOMP IV/35 Central Processor, Modular Computer Systems, Inc., Publication 220-130000-001, April 1977.

[10] MAX IV Basic I/O System, Modular Computer Systems, Inc., Publication 210-610501-000C00, May 1977.

[11] Technical Manual - Data Terminal Model 4805-1, Modular Computer Systems, Inc. Manual is in three sections: publication numbers 225-200125-001 (August 1977), 225-200125-002 (February 1977), and 225-200125-003 (August 1975).

[12] Brown, J. S., Procedures for Building Trend Monitoring System (TMS) MODCOMP Graphics Library and MEGATEK Terminal Program, The MITRE Corporation, WP-6214 (JSC #14826), March 1979.

[13] Functional Design Specification - Trend Monitoring System, NASA Johnson Space Center, JSC #13900, February 1978.

[14] Trend Monitoring System Terminal User's Manual, NASA Johnson Space Center (JSC #14811), 1979.

[15] Diskette Operating System, Data General Corporation, Publication 093-000201-00, August 1976.

[16] Real Time Operating System Reference Manual, Data General Corporation, Publication 093-000056-06, February 1975.

[17] User's Manual - Interface Designer's Reference - NOVA and ECLIPSE Line Computers, Data General Corporation, Publication 015-000031, August 1975.

[18] Nassi, I. and Shneiderman, B., "Flowchart Techniques for Structured Programming." Association for Computing Machinery SIGPLAN Notices, August 1973, pp. 12-26.